Self-Adaptive Fitness in Evolutionary Processes

Thomas Gabor

Dissertation
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig-Maximilians-Universität München

vorgelegt von
Thomas Gabor

eingereicht am
27. Januar 2021

Self-Adaptive Fitness in Evolutionary Processes

Thomas Gabor

Dissertation
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig-Maximilians-Universität München

vorgelegt von
Thomas Gabor

1. Berichterstatter:	Prof. Dr. Claudia Linnhoff-Popien
2. Berichterstatter:	Prof. Dr. Wolfgang Banzhaf
3. Berichterstatter:	Prof. Dr. Jeremy Pitt
Tag der Einreichung:	27. Januar 2021
Tag der Disputation:	6. Juli 2021

Eidesstattliche Versicherung

(siehe Promotionsordnung vom 12. 07. 11, § 8, Abs. 2 Pkt. 5)

Hiermit erkläre ich an Eides statt, dass die Dissertation von mir selbstständig, ohne unerlaubte Beihilfe angefertigt ist.

Thomas Gabor
München, 20. Oktober 2021

Zusammenfassung

Selbst-Adaptive Fitness in evolutionären Prozessen. Die meisten Optimierungsalgorithmen und die meisten Verfahren in Bereich künstlicher Intelligenz können als evolutionäre Prozesse aufgefasst werden. Diese beginnen mit (prinzipiell) zufällig geratenen Lösungskandidaten und erzeugen dann immer weiter verbesserte Ergebnisse für gegebene Zielfunktion, die der Designer des gesamten Prozesses definiert hat. Der Wert der erreichten Ergebnisse wird dem evolutionären Prozess durch eine Fitnessfunktion mitgeteilt, die normalerweise in gewissem Rahmen mit der Zielfunktion korreliert ist, aber auch nicht notwendigerweise mit dieser identisch sein muss. Wenn die Werte der Fitnessfunktion sich allein aus für den evolutionären Prozess intrinsischen Gründen ändern, d.h. auch dann, wenn die extern motivierten Ziele (repräsentiert durch die Zielfunktion) konstant bleiben, nennen wir dieses Phänomen selbst-adaptive Fitness. Wir verfolgen das Phänomen der selbst-adaptiven Fitness zurück bis zu künstlichen Chemiesystemen (*artificial chemistry systems*), für die wir eine neue Variante auf Basis neuronaler Netze entwickeln. Wir führen eine tiefgreifende Analyse diversitätsbewusster evolutionärer Algorithmen durch, welche wir als Paradebeispiel für die effektive Integration von selbst-adaptiver Fitness in evolutionäre Prozesse betrachten. Wir skizzieren das Konzept der produktiven Fitness als ein neues Werkzeug zur Untersuchung von intrinsischen Zielen der Evolution. Wir führen das Muster der Szenarien-Ko-Evolution (*scenario co-evolution*) ein und wenden es auf einen Agenten an, der mittels verstärkendem Lernen (*reinforcement learning*) mit einem evolutionären Algorithmus darum wetteifert, seine Leistung zu erhöhen bzw. härtere Testszenarien zu finden. Wir erkennen dieses Muster auch in einem generelleren Kontext als formale Methode in der Softwareentwicklung. Wir entdecken mehrere Verbindungen der besprochenen Phänomene zu Forschungsgebieten wie *natural computing*, *quantum computing* oder künstlicher Intelligenz, welche die zukünftige Forschung in den kombinierten Forschungsgebieten prägen könnten.

Abstract

Self-Adaptive Fitness in Evolutionary Processes. Most optimization algorithms or methods in artificial intelligence can be regarded as evolutionary processes. They start from (basically) random guesses and produce increasingly better results with respect to a given target function, which is defined by the process's designer. The value of the achieved results is communicated to the evolutionary process via a fitness function that is usually somewhat correlated with the target function but does not need to be exactly the same. When the values of the fitness function change purely for reasons intrinsic to the evolutionary process, i.e., even though the externally motivated goals (as represented by the target function) remain constant, we call that phenomenon self-adaptive fitness. We trace the phenomenon of self-adaptive fitness back to emergent goals in artificial chemistry systems, for which we develop a new variant based on neural networks. We perform an in-depth analysis of diversity-aware evolutionary algorithms as a prime example of how to effectively integrate self-adaptive fitness into evolutionary processes. We sketch the concept of productive fitness as a new tool to reason about the intrinsic goals of evolution. We introduce the pattern of scenario co-evolution, which we apply to a reinforcement learning agent competing against an evolutionary algorithm to improve performance and generate hard test cases and which we also consider as a more general pattern for software engineering based on a solid formal framework. Multiple connections to related topics in natural computing, quantum computing and artificial intelligence are discovered and may shape future research in the combined fields.

Acknowledgments

This thesis is a product of spending almost twelve years at the Institute for Informatics at LMU Munich, from “Analysis for Computer Scientists” to my PhD defense. First of all, I thank Martin Wirsing for new levels of discussion, connecting me with the right people, and providing me with the opportunity to start doing proper science very early on. I also thank Martin Hofmann, who passed on much too soon, and François Bry for inspiring courses, which spoke to me at the right time and nudged me towards research (and taught me about the Curry-Howard isomorphism [78], which to everyone’s surprise happened to come in handy during my thesis defense). Starting out with my Bachelor’s thesis, Matthias Hölzl introduced me to the daily work of a researcher and taught me a lot of skills and wisdom I always aim to pass on to my students as well. He did all that while being a brilliant colleague to work with and thus showing me how much can be done with the right team of right people. I thank him for all of this support.
I later found many of the mentioned right people at the chair for Mobile and Distributed Systems, where I went for my PhD. I want to thank all my past and present colleagues and coworkers there for all the extraordinary times we had doing all kinds of interesting things! I want to express additional gratitude to my colleagues and close personal friends Sebastian Feld, with whom every far-fetched project seemed possible, Steffen Illium, who brought his creativity and all of his heart to the exotic ideas, Thomy Phan, who always knew how to conceive and then execute most ambitious ideas, and Lenz Belzner, who substantially shaped and inspired my scientific journey.
At this point, I would like to again thank Martin Wirsing, who not only got me going as a researcher but also many years later acted as the chair for this thesis’s defense. I would also like to extend my special thanks to my second and third referees for this thesis, Wolfgang Banzhaf and Jeremy Pitt. Both agreed to become a part of this under the most unusual circumstances and it was my great honor to have them and my great pleasure to be talking with them (even *during* my defense), which I can only hope will continue into the future.
Finally and most importantly, I thank Claudia Linnhoff-Popien for providing every opportunity for this long and interesting journey and still continuing to support my ideas and my path as a researcher. At her chair, I was able to have a truly remarkable time and accomplish all the things described in this thesis. I thank her for making possible what no one else could have even foreseen!

Contents

1. Introduction **1**
1.1. Motivation 1
1.2. Research Approach 3
1.3. Structure of This Thesis 5

2. Foundations of Evolutionary Processes **7**
2.1. Formal Framework 7
2.2. Evolutionary Algorithms 9
2.3. Describing Self-Adaptation 11

3. Emergence of Fitness **15**
3.1. Neural Artificial Chemistry Systems 15
3.2. Self-Replication as an Emergent Goal 18

4. The Ideal Fitness **23**
4.1. Diversity-Aware Fitness 23
4.2. Productive Fitness 27

5. Co-Evolutionary Adaptation of Fitness **33**
5.1. Scenario Co-Evolution for Reinforcement Learning 33
5.2. Organizational Patterns for Software Development 37

6. Applications in Natural Computing **41**
6.1. Variants on Evolutionary Algorithms 41
6.2. Quantum Computing 43
6.3. Monte-Carlo Tree Search 44

7. Conclusion **47**
7.1. Summary 47
7.2. Discussion 50
7.3. Outlook 51

Bibliography **53**

A. Papers **61**
A.1. Overview 62
A.2. Credit 64
A.3. Full Texts 71

1. Introduction

[Λέγει ...] τὸν δὲ ἄνϑρωπον ἑτέρῳ ζῴῳ γεγονέναι, τουτέστι ἰχϑύι, παραπλήσιον κατ΄ ἀρχάς.
Humans came into being thoroughly similar to another creature, i.e., a fish, in the beginning.

Anaximander[1]

1.1. Motivation

"Evolution" is used to describe a rather wide rage of phenomena and is thus prone to be one of those terms too general to be useful. We immediately think of biological evolution, which has to a large part shaped the flora and fauna surrounding us using DNA to encode genes, leveraging new gene combinations from random mutations and somewhat guided recombination, and applying natural selection to sort through all that gene information [11, 12]. But we might also talk about the evolution of rock music or fashion or language, when all of these have little to do with the biological processes. Beyond the mere metaphor, a lot of cultural phenomena have been analyzed as an instance of evolution: The term *meme* was coined to resemble the genes of biological evolution in a cultural setting and the process of imitation has been suspected to fulfill the role of biological reproduction [12, 6, 14]. And while the implemented mechanisms are vastly different, similar phenomena can be observed at the macro-level: New specimen arise, find their niche, invade other niches, live together as symbionts or parasites or anything in between, survive, and die out. "Specimen" in this context might range from singular words to whole political systems [14]. In all biological as well as cultural instances of evolution, several (sometimes conflicting) ways "how to do something" arise and fight it out amongst themselves in an open arena. This tends to get rather chaotic so the ultimate outcome can be rather unpredictable.
As put by Dennett [14], evolution can often be regarded as a concept opposite to intentional design. Both approaches try to bring about solutions for a given problem. For intentional design, a final goal is chosen *a priori* and the design process (mostly in a

[1]The Greek original of the quote is indirect speech as Hippolytus in his *Philosophumena* (Book I, Chapter 6) reports in great detail on the philosophy of Anaximander and many others. For ease of reading, this aspect was dropped in the English translation.

top-down approach) implements means to achieve exactly that chosen solution. For evolution, singular small improvements are tested and (in a bottom-up manner) amassed to an overall solution that is then “as good as it got”. Usually, the success of both processes can be evaluated by measuring up the solutions they managed to yield. In design processes, this measurement is often called *quality* or *reward*; evolutionary processes mostly use the term *fitness*. In biological settings, where it is hard to formulate the ultimate problem natural evolution is trying to solve, fitness can hardly be assessed if not *a posteriori*: We assume that a successful gene is one that managed to replicate often; thus a gene that has been replicated often must have had a good fitness [13].
The general field of computer science offers (at least) two interesting takes on that notion of evolution:

- First, the term “software evolution” is widely used to describe the ongoing change in a software ecosystem, even when that change is supposed to be intentionally designed top-down in its entirety. We could see the usage of the term “evolution” here as a wise capitulation in front of the sheer number of players and possible developments in a larger software ecosystem.[2] Somewhat recent research in software engineering might be a sign of an upcoming paradigm shift on multiple fronts [62, 63]. We also point out a few arguments following that direction, including a unification of both the biological and the software engineering notion of evolution, which is why we return to that discussion in Chapter 5.

- Second, computer scientists have been building artificial, virtual instances of evolution and used them for their own purposes. Most of them fall into the category of “evolutionary algorithms” and for most of them their purpose is optimization. Please note that there also are some purely un-guided instances of artificial evolution and they will make a cornerstone of this thesis (cf. Chapter 3). Evolutionary algorithms, however, are usually given a certain, pre-defined, fully specified, intentional goal. This goal is commonly passed to the evolutionary algorithm via an *a priori* defined fitness function and from there on the evolution works its magic. What is happening exactly when we just pass in a goal as fitness is the main topic of Chapter 4.

In both of these instances of evolution in software, a process of evolution is deliberately put in place to fulfill a purpose envisioned by some system designer. In the instance of evolutionary algorithms, we see it most clearly: They are used as optimizers for a given fitness function. By contrast, for natural (or even cultural) evolution, it is largely assumed that the information units themselves are the main beneficiaries of the process of evolution (compare Dawkin’s notion of “selfish genes” [12]). But evolutionary algorithms are programmed for the good of some other external entity. And they still work!

[2]That only applies in a context where the term is strongly connected with its biological meaning, however. Curiously, the word *evolution* originates from Latin *ēvolvere*, which literally means “to roll out”—oddly fitting for a series of software releases.

In this thesis we build on the observation made by many researchers and practitioners that evolutionary processes that are simply told a goal via a fitness function do not necessarily make the best decisions with respect to that fitness function along the way [18, 49, 3, 17, 56, 76, 73]. We can also turn that observation the other way around: There are experiments where evolutionary algorithms return better results with respect to a fitness function f if they are told to optimize for a different fitness function $f' \neq f$. Metaphorically, we might imagine that any evolutionary process still retains some of its natural selfishness, some of its own inherent goals. Of course, we will describe this phenomenon much more precisely throughout the main parts of this thesis.
Changing goals (and as a consequence changing fitness) has already been an extensive topic of research on evolutionary algorithms and among other names has been called *adaptive fitness* [40, 45, 82, 21]. For the titular *self-adaptive fitness* we assume that the fitness used within an evolutionary algorithm is supposed to change even when its externally given purpose remains the same. We derive the *why* from examining artificial chemistry systems without any given fitness that still show emergent fitness properties (cf. Chapter 3) and the transformation of a given fitness function into the *productive fitness* (cf. Chapter 4). We also show the *how* by discussing diversity and co-evolution in evolutionary processes (cf. Chapters 4 and 5).

1.2. Research Approach

We do provide some contributions on formal methods, providing a sound, encompassing formal framework for any description of evolutionary processes and related phenomena [32, 33]. Most of the provided results, however, are empirical and based on *in silico* experiments, from which we draw our conclusions. We use this method throughout the entire analysis of self-adaptive fitness in evolutionary processes. The main direction of research is guided by the following research questions (RQs).

(RQ1) Which phenomena can be described by evolutionary processes? As hinted in the introduction, evolution or evolutionary processes are broadly defined terms, applying both to artificially constructed processes as well as natural phenomena. We provide a formally sound framework and give precise definitions for all the mathematical tools we need.

(RQ2) Which phenomena can be described as self-adaptive fitness? There is the clear case of self-adaptive fitness where the fitness function used by the individuals (the subjective fitness) features some free parameters and these parameters are left to change due to the evolutionary process. We analyzed this case in great detail for the example of diversity-aware fitness functions [28, 29].[3] We noticed a potential benefit

[3]Diversity-aware fitness functions are self-adaptive because their diversity part depends on the whole population, which changes throughout evolution per definition.

in both instances where the fitness function given as part of the problem definition (the objective fitness) changed during evolution as well as instances where the objective fitness was not changed at all. The fact that one can benefit from a changing fitness in an unchanging environment especially sparked our interest and led into the next research question.

(RQ3) Are there goal-independent parts to fitness? Research led directly into the field of artificial chemistry systems: These are interaction systems based on particles which evolve via the application of predefined interaction rules without any specific goal. However, under the right circumstances, these systems exhibit a natural drive towards some form of stability. While usually defined on simpler particle data types, we showed how neural networks can be used as particles while still showing similar convergence behavior. We have thus confirmed that fitness goals may exist even when not explicitly given to the algorithm through its environment. Inspired by these experiments, we can ask the next research question.

(RQ4) Is there some inherent goal fitness in evolutionary algorithms as well? Evolutionary algorithms are usually given an objective goal fitness or otherwise tend to end in total chaos.[4] But if they work better when given variations of the objective goal as subjective fitness, they do seem to follow a slightly different goal of their own. We have defined the notion of productive fitness to describe that phenomenon and have shown that subjective fitness functions that approximate productive fitness better also yield better overall results in evolutionary algorithms. Sadly, productive fitness comes with the major disadvantage that we can only compute or even accurately approximate it a posteriori.

(RQ5) How can we discover goal functions for evolutionary processes to follow? We could put this differently: When I am given an objective goal to solve, which subjective goal should I give to my evolutionary algorithm to match? Or simply: How do I find the right goal function? Of course, the evolutionary scientist's answer to such search problems is to use an evolutionary algorithm. Co-evolution is a well-researched method where multiple populations (of different types or with different goals) evolve side by side and thereby influence each other, usually being coupled by their fitness evaluation. It is closely related to adversarial learning, which is widely used in many variants of reinforcement learning. We showed that a reinforcement-learning-based agent can be coupled with an evolutionary algorithm to generate the currently right goals for the learning process of the agent.

[4]We can suspect that typical crossover-based recombination functions breaking up schemes is the main reason for that [75]. If you replace recombination with an allele-wise average operator in evolutionary algorithms, you tend to notice convergence much more easily.

(RQ6) Are there similar phenomena in other kinds of evolutionary processes? Having become acquainted with relaying information from the objective goal in various ways to changing subjective goals, we researched various methods in natural computing and have come up with some further variations on evolutionary algorithms, new analyses regarding quantum annealing, and a new approach for Monte-Carlo tree search. Of course, this research question is not to be answered completely any time soon.

This thesis sums up all the described research relevant to these questions. The process of the conducted research skips over more well researched topics (artificial chemistry systems, adversarial learning, e.g.) to get to more specific new discoveries (neural artificial chemistry, evolutionary-RL hybrid learning, respectively). We feel that such a course of action is fitting for research on a fundamental phenomenon as fitness, cross-cutting with many fields in natural computing, artificial intelligence, and optimization.

1.3. Structure of This Thesis

The body of this thesis in large parts follows the course of questions (and corresponding actions) sketched in the previous section. We repeat larger parts of the formal model defined in [33, 35] as a common formal foundation, defining basic notions such as evolutionary process, fitness, or objective fitness for all work in this thesis. For the other main chapters, we provide a very brief introduction to the relevant fields but refer to literature for a complete understanding. In the body of this thesis, we only give the main results and refer to the attached papers for the way to get there. The main body of this thesis is structured as follows:

thesis chapter	research questions	main papers
2. Foundations of Evolutionary Processes	RQ1, RQ2	[33]
3. Emergence of Fitness	RQ3	[31]
4. The Ideal Fitness	RQ2, RQ4	[26, 28, 29, 35]
5. Co-Evolutionary Adaptation of Fitness	RQ5	[27, 32, 36, 37]
6. Applications in Natural Computing	RQ6	[25, 68, 38, 30, 34]

Table 1.1.: Overview of the contents of this thesis.

Section 7 provides an overall conclusion and sketches some directions of ongoing and future research. For the sake of brevity, we only cite necessary references in the thesis and refer to the attached papers for a more in-depth review of related work. After the bibliography, we provide a rich appendix that contains additional description of all main papers listed in Table 1.1 as well as reprints of the full texts according to LMU Munich requirements.

2. Foundations of Evolutionary Processes

[Οὗτος ἔφη ...] καὶ τὰ πάντα φύεσθαί τε καὶ ῥεῖν τῇ τοῦ πρώτου ἀρχηγοῦ τῆς γενέσεως αὐτῶν φύσει συμφερόμενα.
All things evolve and flow corresponding in nature with the first initiator of their development.

Thales[1]

2.1. Formal Framework

We understand evolution as a process that takes one of several possible paths; that choice, however, is built iteratively by choosing but one step at a time. These steps usually imply some sort of locality, be it in time or state, in contrast to planning, e.g., where a number of steps are taken usually with a superior goal being the ultimate reason for every single choice.

Definition 1 (Evolution[2])**.** *Let $\mathcal{X}$ be an arbitrary set called* state space*. Let $g \in \mathbb{N}$ be called the* generation count*. Let $X_i \subseteq \mathcal{X}$ for any $i \in \mathbb{N}, 0 \leq i \leq g$, be a subset of $\mathcal{X}$ called* population*. Let $E : \mathfrak{P}(\mathcal{X}) \to \mathfrak{P}(\mathfrak{P}(\mathcal{X}))$ be a function called* evolutionary function*. A tuple $(\langle X_i \rangle_{0 \leq i \leq g}, E)$ is called an* evolution *over $\mathcal{X}$ iff $X_i \in E(X_{i-1})$ for all $i \in \mathbb{N}, 1 \leq i \leq g$.*

At this level of generality, the notion of evolution can be applied to phenomena in various domains: If $\mathcal{X}$ is the space of all possible genetic codes and E describes the biological processes in the physical world, biological evolution fits Definition 1. If $\mathcal{X}$ is the space of all possible software artifacts and E describes a common development process, software evolution fits Definition 1 as well. Note that in Definition 1 we do imply a temporal ordering of simple steps following each other, leading to some sort of locality in time

[1]The Greek original of the quote is (again) indirect speech as Hippolytus (*Philosophumena*, Book I, Chapter 1) reports on Thales of Miletus.

[2]The definition given here (as most definitions in this chapter) closely resembles the definitions given in [33, 35], which originate from the formalization in [36]. Please note that these definitions are designed to work mathematically for what we need in the following study while encapsulating as many phenomena as possible semantically.

(i.e., the evolutionary function will not have an effect randomly popping out a hundred generations later; everything it does needs to be represented in its direct output, which is the next generation). However, we refrain from enforcing any sort of locality in the state space by our definitions, as these locality properties then need to be defined for every single state space, costing us generality in the end.
Stemming from various assumptions of gradual, local change, evolution in nature is commonly described as "blind" [13], i.e., there is no discernible goal that biological evolution, e.g., is actively heading towards as a whole.[3] Still, even natural evolution certainly follows certain trends, especially when we let it play out in rather stable settings. Even more so, processes like software evolution are filled with specified goals, which might or might not influence where the development process is actually headed.
Formally, we can define a target to an evolution as a predicate on the population and can then check if an actual evolution matches our specified target. Since we not only want a binary "target hit" vs. "target not hit" distinction, we use the following definition w.l.o.g.:

Definition 2 (Target [33]). *Let $\mathcal{X}$ be a state space. A function $t : \mathcal{X} \to [0;1]$ that assigns to all elements in the search space a scalar value is called a* target function*.*

Note that we define a target to be a function on the state space only. It allows us to formally capture where an evolution currently is (i.e., which points and areas of the search space it occupies) and where we would want it be (for any kind of reason), regardless of how it got there or where it could have gone.[4] For artificial evolutions (like software evolution), a certain target to be achieved is usually the reason the whole evolution was designed in the first place. Even for natural evolution, biologists have spent a long time researching what underlying targets there might exist: "Survival of the fittest" is most commonly used to describe the target where evolution is headed [52]. Dawkins [13] gives a more nuanced explanation that describes the replication of information patterns (dubbed "genes" there) as the ultimate target of evolution. In any case, an understanding of evolution is based on understanding the target it is headed, even when that goal is never explicitly represented.
We use the notion of an evolutionary process to mean an evolution together with a given target function that it can be evaluated against.

Definition 3 (Evolutionary Process [33]). *Let $\mathcal{X}$ be a state space. Let $E : \mathfrak{P}(\mathcal{X}) \to \mathfrak{P}(\mathfrak{P}(\mathcal{X}))$ be an evolutionary function. Let $t : \mathcal{X} \to [0;1]$ be a target function. Let X_i be a population for any $i \in \mathbb{N}, 0 \leq i \leq g$. A tuple $\mathcal{E} = (\mathcal{X}, E, t, \langle X_i \rangle_{i \leq g})$ is an* evolutionary process *iff $(\langle X_i \rangle_{i \leq g}, E)$ is an evolution.*

[3]Note the "active" part here, implying that even when populations evolve towards certain targets, they usually have no means to be aware of that targets.

[4]Obviously, various means to measure the difference between these targets might be defined. Given a target function t, we might for now just image a simple formula like $\sum_{x \in X} t(x)$ to describe how well a population X matches the target function t.

The target function is what we evaluate the success of the whole evolutionary process against. Usually, we want an evolutionary process to maximize $\max_{0 \leq i \leq g} \max_{x \in X_i} t(x_i)$ or some similar measurement. To do so, we usually evaluate single individuals during the course of evolution as well in order to steer the whole process. The value of the evaluation of a single individual at a specific step in evolution is called *fitness*. The fitness function can be a target function or even be equal to the evolutionary process's overall target function t; however, it does not need to be equal or even close to the target function t given to the whole process. To discern the two, we call the target function used to evaluate the evolutionary process as a whole the *objective target* or *objective fitness* and, when there is any need for clarification, any target function used to evaluate the individuals is by contrast called *subjective target* or *subjective fitness*. In contrast to the target function, a fitness function can be parametrized on more than just the state space: It can reflect the current state of the evolution as well as the evolutionary process's complete history, which means that for a single point of the state space it can change over time. We write $\mathfrak{E}$ for the space of all evolutionary processes according to Definition 3. Furthermore, we allow fitness to change for no reason at all. Formally, we model this by passing on a random parameter from a space of random parameters $\mathfrak{R}$.[5]

Definition 4 (Fitness [33])**.** *Let $\mathcal{X}$ be a state space. A function $f : \mathcal{X} \times \mathfrak{E} \times \mathfrak{R} \to [0; 1]$ is called a* fitness function*. This function takes an individual, its evolutionary process up until now, and random input and returns a scalar value.*

Note again that the definition of a fitness function does not necessarily imply that any evolutionary process is actually pursuing it. Usually, a fitness function is used in some way to determine the possible outcomes given by the evolutionary function E of an evolutionary process $\mathcal{E}$.
The definition of evolutionary processes given here is general enough that it can subsume artificial chemistry systems (discussed in Chapter 3) as well as evolutionary algorithms (our main object of research for Chapters 4 and 5 as well es Section 6.1).

2.2. Evolutionary Algorithms

As evolutionary algorithms are the topic of large parts of this thesis, we shall introduce them formally to a bit greater detail, still following the construction in [33]. Naturally, we define an evolutionary algorithm as a special case of an evolutionary process, where the evolutionary function E is of a certain form. To construct said form, we first introduce the notion of *evolutionary operators*. Most commonly, evolutionary algorithms use three distinct types of operators [33]:

Mutation. This operator $mut : \mathcal{X} \times \mathfrak{R} \to \mathcal{X}$ generates a randomly slightly altered individual from a parent.

[5]This resembles how pseudo-random number generators work in most computers. However, we will not regard true randomness as an issue here [88].

Recombination. This operator $rec : \mathcal{X} \times \mathcal{X} \times \mathfrak{R} \to \mathcal{X}$ takes two individuals to combine them into a new individual.

Migration. This operator $mig : \mathfrak{R} \to \mathcal{X}$ generates a random new individual.

As we did previously, we use $\mathcal{X}$ for the state space and $\mathfrak{R}$ for the space of random numbers, allowing these operators to operate non-deterministically (when we do not know the random number they are given). We further usually write the random parameter as a subscript to the function call, i.e., $mut_r(x)$ mutates individual x using a random (projection of) a random number r. Note that we treat selection differently as it is usually put as a preprocessing step before the application of some operators. We adopt the following definition:

Definition 5 (Selection)**.** *A function $s_k : \mathfrak{P}(\mathcal{X}) \times \mathfrak{E} \times \mathfrak{R} \to \mathfrak{P}(\mathcal{X}^k), k \in \mathbb{N}$ is called a* selection function *iff $s_k(X, \mathcal{E}, r) \subseteq X^k$ for all $X, \mathcal{E}, r$. This function takes a population, its evolutionary process, and random input and returns a subset of vectors of length k consisting of individuals from the given population.*

As a bit of notational sugar, we introduce a short-hand notation for the standard map function from functional programming: We write $op_r\langle X\rangle =_{def} \bigcup_{x\in X} op_r(x)$ for the set of all results of applying the operator op to all elements of X. We also introduce the short-hand $op_r\langle i\rangle = \bigcup_{i\in\mathbb{N},0\leq i<m} op_r()$ to simply construct a set of multiple outputs of $op_r()$.[6]

Definition 6 (Evolutionary Algorithm)**.** *Let s_1^{mut}, s_2^{par}, and s_1^{sur} be suitable selection functions. Let $m \in \mathbb{N}$ be the number of migrants. An evolutionary process $\mathcal{E} = (\mathcal{X}, E, t, \langle X_i\rangle_{i\leq g})$ is called an evolutionary algorithm iff the evolutionary function E is of the form*

$$E(X) = \bigcup_{r\in\mathfrak{R}} s_1^{sur}\Big(X \cup mut_r\langle s_1^{mut}(X, \mathcal{E}, r)\rangle \cup rec_r\langle s_2^{rec}(X, \mathcal{E}, r)\rangle \cup mig_r\langle m\rangle\Big).$$

For an evolutionary algorithm $\mathcal{E}$, we also further assume (unless stated otherwise)

- that s_1^{sur} is called *survivor selection*, its result is mostly based on the individuals' fitness, and $|s_1^{sur}(X)| = |X_0|$ for all $X \subseteq \mathcal{X}$, i.e., the population size remains constant,

[6] A quick note on the details: First, in this chapter we commonly operate on multisets, meaning that two individuals can be identical and still exists within the same population, for example. Furthermore, the encoding of randomness requires us to either allow for the random number r that is passed to all operators to be set to a different value in between calls or for each operator to not be a pure function and instead be able to choose a different projection of parts of r (which we imagine to be a really long number). We decided to remain agnostic to a specific model of randomness, so we just carry the random parameter throughout the signature but do not further explain how to use it.

- that s_1^{mut} is a random selection of a subset of the given population, with the size of the subset being determined by applying a mutation rate to every individual,
- that s_2^{par} is called *parent selection* and that the size of its selection is determined by applying a recombination rate with a bias towards individuals with higher fitness.

Further note that the model of evolutionary algorithms is not general enough to encompass all variants found in literature. Most prominently, we do not have in-place recombination and not even in-place mutation, which means that any newly generated individuals will be added to the population without directly kicking someone else out. This means that we only need to make survivor selection s_1^{sur} elitist for the whole evolutionary algorithm E to be elitist.[7] Furthermore, we fixed two-parent recombination, the independence of mutant offspring and recombination offspring and more minor properties of evolutionary algorithms that may be treated differently in literature. However, Definition 6 is general enough that we can use it for all occurrences of evolutionary algorithms in this thesis. For more details, again, see [33].
Furthermore, we provide a software package that implements evolutionary algorithms in the shape of our formal framework in Python and is available to download.[8] It was used for the research presented in [26, 28, 29, 33, 35].

2.3. Describing Self-Adaptation

Adaptation in its own right has been a target of many fields of research and many attempts at a formalization of the notion exist [48, 67, 66], both from an intensional and an extensional perspective. Difficulty usually arises when we try do discern adaptive behavior of a system from non-adaptive behavior as every *single* behavior might always be simply hard-coded into the system and should then probably not be called adaptive at all. Note that when we talk of "systems" and "behavior" in this context, we again mean it in the most general way possible. Hölzl and Wirsing [48] give the following definition, which already we repeat in [32, 37]:

Definition 7 (System [48]). *Let I be a (finite or infinite) set, and let $\mathcal{V} = (V_i)_{i \in I}$ be a family of sets. A* system *of type $\mathcal{V}$ is a relation S of type $\mathcal{V}$.*

Definition 8 (Composition [32, 37]). *Let S_1 and S_2 be systems of types $\mathcal{V}_1 = (V_{1,i})_{i \in I_1}$ and $\mathcal{V}_2 = (V_{2,i})_{i \in I_2}$, respectively. Let $\mathcal{R}(\mathcal{V})$ be the domain of all relations over $\mathcal{V}$. A* combination operator *$\otimes$ is a function such that $S_1 \otimes S_2 \in \mathcal{R}(\mathcal{V})$ for some family of sets $\mathcal{V}$ with $V_{1,1}, ..., V_{1,m}, V_{2,1}, ..., V_{2,n} \in \mathcal{V}$. The application of a combination operator is called* composition*. The arguments to a combination operator are called* components.

[7]Elitism means that the best value of a population only gets better over time and makes some parts of analyzing an evolutionary algorithm much easier. For a more in-depth discussion on elitism, especially in the context of potentially varying fitness, please see [33].
[8]Please see `github.com/thomasgabor/isola-evolib`.

Note that we are talking about arbitrary sets and relations over them as systems and we just define that we can put them together to form new systems. Hölzl and Wirsing [48] further assume that there is a space of predicates and that we can use these predicates to make Boolean propositions about a system S. We write $S \models \gamma$ if the predicate γ resolves to *true* when applied to the system S.
We augment that model to include not only binary predicates, but also (potentially real-valued) evaluation functions [32, 37].[9] These evaluation functions are in fact target functions if we assume that the system space $\mathcal{V}$ is the search space $\mathcal{X}$ of the target function.
We can then add some more definitions to the framework, which then allow us to repeat the definition for adaptation originally given by [48] for systems with target functions. Please note that formally, environments are just systems.[10] The distinction is purely aesthetic.

Definition 9 (Adaptation Domain [32, 37])**.** *Let S be a system. Let $\mathcal{H}$ be a set of environments that can be combined with S using a combination operator $\otimes$. Let Γ be a set of Boolean goals. Let F be a set of evaluation values and $\preceq$ be a preorder on F. Let Φ be a a set of evaluation functions with codomain F. An* adaptation domain *$\mathcal{A}$ is a set $\mathcal{A} \subseteq \mathcal{H} \times \Gamma \times \Phi$. S can adapt to $\mathcal{A}$, written $S \Vdash \mathcal{A}$ iff for all $(H, \gamma, \phi) \in \mathcal{A}$ it holds that $S \otimes H \models \gamma$.*

Definition 10 (Adaptation Space)**.** *Let $\mathcal{H}$ be a set of environments that can be combined with S using a combination operator $\otimes$. Let Γ be set of goals. Let Φ be a a set of evaluation functions. An* adaptation space *$\mathfrak{A}$ is a set $\mathfrak{A} \subseteq \mathfrak{P}(\mathcal{H}, \Gamma, \Phi)$.*

Definition 11 (Optimality [32, 37])**.** *Given two systems S and S' as well as an adaptation space $\mathcal{A}$, S' is at least as optimal as S, written $S \preceq_{\mathcal{A}} S'$, iff for all $(H, \gamma, \phi) \in \mathcal{A}$ it holds that $\phi(S \otimes H) \preceq \phi(S' \otimes H)$.*

Definition 12 (Adaptation [32, 37])**.** *Given two systems S and S', S' is at least as adaptive as S with respect to optimization, written $S \sqsubseteq^* S'$ iff for all adaptation domains $\mathcal{A} \in \mathfrak{A}$ it holds that $S \Vdash \mathcal{A} \Longrightarrow S' \Vdash \mathcal{A}$ and $S \preceq_{\mathcal{A}} S'$.*

We can see that this notion of adaptation is relative, i.e., we can only *compare* two systems regarding their adaptivity but never retrieve an objective measurement. Further note that the Definition 12 is purely extensional: More adaptation means more environments where we can uphold the predicates and achieve at least the same evaluation result. This might be achieved by means that, intuitively, we would in no way call adaptive.
Nonetheless, we can use Definition 12 to define the more intuitive notion of adaptivity as change over time.

[9] We already call these functions *fitness function* in [32, 37], but substitute the term here for better notational consistency.

[10] Further note that we change the notation for environments for consistency with our above definitions and slightly adjust the vocabulary used in Definitions 9–13 for the same reason.

Definition 13 (Adaptation Sequence [32, 37]). *A series of $|I|$ systems $\mathcal{S} = (S_i)_{i \in I}$ with index set I with a preorder $\leq$ on the elements of I is called an* adaptation sequence *iff for all $i, j \in I$ it holds that $i \leq j \implies S_i \sqsubseteq^* S_j$*

We can now see where our two strains of definitions come together: An evolutionary process that is elitist w.r.t. the target function t is an adaptation sequence for (at least) adaptation domains with $\Gamma = \emptyset$ and $\Phi = \{t\}$. We return to that connection in Chapter 5, when we construct a design pattern from co-evolving adaptation sequences.
In [32, 37] we end up with a rather technical definition for self-adaptation as a special case of an adaptation sequence:

Definition 14 (Self-Adaptation [32, 37]). *A system S_0 is called* self-adaptive *iff the sequence $(S_i)_{i \in \mathbb{N}, i < n}$ for some $n \in \mathbb{N}$ with $S_i = S_0 \otimes S_{i-1}$ for $0 < i < n$ and some combination operator $\otimes$ is an adaptation sequence.*

Intuitively we can read that as a system that is able to adapt to its own current state; or simply as a system that forms an evolutionary process (even when the circumstances remain identical). This is a rather broad definition meant to encapsulate a variety of different phenomena.
So what do we mean when we apply the notion "self-adaptive" to the concept of fitness as we ask in the title of this thesis? We mean an evolutionary process with a fitness that changes on its own accord, triggered perhaps by outside events but also by events originating from within the evolutionary process only.

3. Emergence of Fitness

[Νομίζει τὰς οὐσίας ...] στασιάζειν δὲ καὶ φέρεσθαι ἐν τῷ κενῷ διά τε τὴν ἀνομοιότητα καὶ τὰς ἄλλας τὰς εἰρημένας διαφοράς [...].
[Particles] quarrel and drift through empty space because of their inhomogeneous form and their further mentioned differences [of shape, magnitude, ...].

Democritus[1]

3.1. Neural Artificial Chemistry Systems

Let us back up a bit: In the previous chapter, we have given some formal definitions for phenomena such as fitness, evolutionary algorithms, and adaptation. In this chapter we return to a more rudimentary notion of an evolutionary process. Our main question is: Would fitness still exist if we did not explicitly give a fitness function for an evolutionary process? For biological evolution, the answer is clear: No one ever explicitly gave a fitness function for this evolutionary process and still biology observes the fitness of certain information patterns (individuals, behaviors, etc.) within evolution [13].
So what can we do with an evolutionary process without any given goals? What remains is actually pretty close to a formal construct called *artificial chemistry system*. Although most common definitions of that notion read a bit differently [16], we shall define it in the context of the definitions made in Chapter 2. To this end, we first define two evolutionary operators:

Live. This operator *live* : $\mathcal{X} \times \mathfrak{R} \to \mathcal{X} \cup \{\emptyset\}$ generates an altered individual from a given individual.

Meet. This operator *meet* : $\mathcal{X} \times \mathcal{X} \times \mathfrak{R} \to (\mathcal{X} \cup \{\emptyset\}) \times (\mathcal{X} \cup \{\emptyset\})$ takes two individuals and returns two (possibly) altered individuals.

[1]Here, Simplicius of Cilicia describes Democritus's philosophy in his commentary of Aristotle's work "Περὶ οὐρανοῦ" ("On the Heavens"). Again the indirect speech is removed in translation and some liberty is taken to add information from the context of a rather lengthy description of atomic objects, which can be naturally equated with particles when talking about artificial chemistry.

For the sake of simplicity, we make a few choices here that are peculiar for artificial chemistry systems:

- At most two individuals can be involved in the application of an evolutionary operator.[2]
- Since these operators cannot produce more individuals than they consume, the amount of individuals in a system is limited to the amount we started with. However, individuals may die out when either operator returns an empty value $\emptyset$ in place of an individual. The operator might instead also be defined to simply generate a new random particle and never return $\emptyset$.

We further provide a predicate $\mathit{done}_X : \mathcal{X} \to \mathbb{B}$ so that for any individual x in the population X, when x is involved as an argument in the application of an evolutionary operator, it holds that $\mathit{done}(x)$ is true and if x has not been used in an evolutionary operator, $\mathit{done}(x)$ is false. We use this predicate to model change in individuals via adding new individuals and having them replace the old.[3] Obviously, we need to "reset" the predicate when a new population is generated (i.e., after one computation of the evolutionary function $X_{t+1} = E(X_t)$ at time t). Furthermore, we provide a fixed selection function $s_1^{\mathit{done}}(X, \_, \_) = \{x \in X \mid \neg \mathit{done}_X(x)\}$.

Definition 15 (Artificial Chemistry System, Soup)**.** *Let s_1^{live}, s_2^{meet} be suitable selection functions. An evolutionary process $\mathcal{E} = (\mathcal{X}, E, t, \langle X_i \rangle_{i \leq g})$ is called an artificial chemistry system or (synonymously) a soup iff the evolutionary function E is of the form*

$$E(X) = \bigcup_{r \in \mathfrak{R}} s_1^{\mathit{done}} \Big(X \cup \mathit{live}_r \langle s_1^{\mathit{live}}(X, \mathcal{E}, r) \rangle \cup \mathit{meet}_r \langle s_2^{\mathit{meet}}(X, \mathcal{E}, r) \rangle \Big).$$

Even though far more intricate artificial chemistry system can be defined, we further assume for any artificial chemistry system $\mathcal{E}$ we consider in this thesis (unless stated otherwise)

- that s_1^{live} selects all individuals for automatic change over time (i.e., "growing" or "aging"), i.e., $s_1^{\mathit{live}}(X, \mathcal{E}, \_) = X$.
- that s_2^{meet} selects random pairs of individuals, i.e., that any individual has the same chance of meeting any other individual, and that chance is called *meeting rate.*

[2]In the common wording for artificial chemistry systems, this means we allow at most binary interaction rules that also return at most two particles.

[3]This is usually implemented by adding a unique ID to individuals and removing all identical IDs from the population multi-set when adding an new individual. We could also reserve a certain subspace of $\mathcal{X}$ for "deprecated" individuals and simply move the arguments of operators there during application. Practically, this results in the same behavior. We chose the following notation for its similarities to Definition 6.

In literature, lots of different types of individuals and their respective interactions (i.e., evolutionary operators) have been considered, ranging from simple bitstrings ($\mathcal{X} = \mathbb{B}^n$) to complex representations of algorithms (where $\mathcal{X}$ might be the space of all state automata or Turing machines [16, 2]). We consider the findings of Fontana and Buss [24] to be most interesting for the emergence of goals in evolution: They use individuals that are arbitrary λ-expressions in the λ-calculus [43]; one step of β-reduction and the application operation "_ _" for two λ-expressions then lend themselves naturally for *live* and *meet*, respectively. We will get into their results in a bit more detail in the following section.

For this thesis, we additionally examine the use of neural networks as individuals in an artificial chemistry system. Neural networks are widely used as universal function approximators and can thus also possibly encode rather complex behavior for particles in an artificial chemistry system. We suggest that in future work neural network soups might be integrated with machine learning approaches for which neural networks are vital nowadays. For a full introduction to neural networks, we refer to the ample literature on that topic [20, 55] or to the formally rather complete definition we give in [31].

For now, we assume that a neural network encodes a function $\mathcal{N} : \mathbb{R}^p \to \mathbb{R}^q$ for input length p and output length q via a vector of r weights $\overline{\mathcal{N}} \in \mathbb{R}^r$. By the construction of neural networks, it holds that $r > p$ (and usually $r \gg p$ if the networks perform a complex task). Thus, neural networks cannot easily process other neural networks of the same structure and size. From an analogous argument it follows that neural networks also cannot *produce* other neural networks of the same structure and size. We present workarounds for that in [31] so that we can write $\mathcal{O} = \mathcal{N} \lhd \mathcal{M}$ to mean that $\mathcal{O}$ is the neural network that is generated as the output of the neural network $\mathcal{N}$ when given the neural network $\mathcal{M}$ as input even where $|\overline{\mathcal{M}}| = |\overline{\mathcal{N}}| = |\overline{\mathcal{O}}|$, i.e., the neural networks $\mathcal{M}$, $\mathcal{N}$, and $\mathcal{O}$ share the same structure and only differ in the value of their weights. In [31] we further provide means that allow us to write $\mathcal{O} = \mathcal{N} \leftrightsquigarrow \mathcal{M}$ to mean that $\mathcal{O}$ is the neural network that is generated by training the neural network $\mathcal{M}$ to exactly reproduce the weights of a neural network $\mathcal{N}$ when given said weights as input (with one step of training each).

This framework now allows us to define two evolutionary operators as instances of *live* and *meet* designed specifically for neural networks as individuals, i.e., $\overline{\mathcal{N}} \in X \subseteq \mathcal{X}$.[4]

Self-Train. This operator is an instance of *live* : $\mathcal{X} \times \mathfrak{R} \to \mathcal{X} \cup \{\emptyset\}$ and introduces a hyperparameter $A \in \mathbb{N}$ to configure the intensity of its application. It is defined as

$$\textit{self-train}_r(\mathcal{N}) = \mathcal{N} \underbrace{\leftrightsquigarrow \mathcal{N} ... \leftrightsquigarrow \mathcal{N}}_{A \ times}.$$

[4]In the following definitions and more, we will neglect converting a network's functional representation $\mathcal{N}$ to its weight representation $\overline{\mathcal{N}}$ and vice versa as it is always trivial in the cases we discuss, where network structure is never altered. We still keep both notations for consistency with [31].

Attack. This operator is an instance of $meet\colon \mathcal{X} \times \mathcal{X} \times \mathfrak{R} \to (\mathcal{X} \cup \{\emptyset\}) \times (\mathcal{X} \cup \{\emptyset\})$. It is defined as

$$attack_r(\mathcal{M}, \mathcal{N}) = \mathcal{N} \lhd \mathcal{M}.$$

As we show in [31], these operators allow us to instantiate artificial chemistry systems that exert interesting properties. We focus on our main observation in the following section.

3.2. Self-Replication as an Emergent Goal

As we have shown with Definition 15, artificial chemistry systems can be regarded as an evolutionary process in accordance with Definition 3; yet one slot remains empty, i.e., we have never explicitly specified any target function. Furthermore, none of our evolutionary operators and nothing in the evolutionary function E incorporates any kind of fitness measurement. Still it is clear that (only going by the evolutionary operators) some individuals are more likely to survive than others. Let us look at a few cases.

Real Number Soup. We start with a clear cut example. We consider an artificial chemistry system (or shorter: soup) $\mathcal{R}$ with a population made up of real numbers between -1 and 1, i.e., $\mathcal{X} = [-1; 1] \subseteq \mathbb{R}$, initially sampled at random (uniformly). We define a clipped multiplication operator $\boxdot : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ as

$$x \boxdot y = \begin{cases} -1 & \text{if } x \cdot y < -1 \\ 1 & \text{if } x \cdot y > 1 \\ x \cdot y & \text{otherwise.} \end{cases} \tag{3.1}$$

We then use $live_r(x) = x \boxdot (1 + y)$ where y is random uniform sampled from $\mathcal{X}$ and $meet_r(x, y) = x \boxdot y$. We quickly see that the population of this evolutionary process $\mathcal{R}$ is approaching 0 over time: There might be some movements away from 0 during *live* and even jumps over 0 when a positive and a negative number *meet*, but in general (given the rest of the evolutionary function is defined somewhat reasonably) the population will converge rather quickly to this trivial goal. Furthermore, once arrived at near 0, the effects of the evolutionary operators diminish completely; so there really is no escape.
This example is, of course, rather trivial but shows that a goal for the process has emerged solely from the employed operators, even though the operators are applied fully randomly. In this case, it is a property of multiplication that comes through and defines the overall target that the evolutionary process will (most likely) end up at.

λ-Expression Soup. Fontana and Buss [22, 23] provide an example where the evolutionary process's implicit target has a much more real computational meaning. As briefly mentioned earlier, they describe a soup $\mathcal{L}$ where the search space $\mathcal{X}$ is the space

of all valid λ-expressions in normal form. Thus, our population might be made up of individuals such as $(\lambda x.\ x)$ or $(\lambda x.\ (\lambda y.\ y\ x))$. In this case we make no use of *live* and define $\mathit{meet}(x, y) = (x, x\ y)$, i.e., when two particles meet one of them becomes the λ-application of the two.[5] Since the search space is restricted to normal form λ-expressions, we need to apply β-reduction to the result of each operation.[6] Without further constraints and conditions, Fontana and Buss observe the emergence and dominance of "single self-copying functions or ensembles of hypercyclically coupled copying functions' [22].[7] The most basic example of such functions is $(\lambda x.\ x)$ but they also find more complex structures [23]. Again, these structures arise naturally without us steering the evolution using fitness or biased selection.
Of course, the structures that arise are not very random at all. Self-copying functions like $(\lambda x.\ x)$ form a fixpoint of the *meet* operator:[8]

Definition 16 (Fixpoint)**.** *Let $x \in \mathcal{X}$ be an individual of an artificial chemistry system $\mathcal{E}$. The individual x is called a fixpoint of $\mathcal{E}$ iff $\mathit{meet}(x, x) = (x, x)$.*

We can easily verify that

$$\mathit{meet}((\lambda x.\ x), (\lambda x.\ x)) = ((\lambda x.\ x), (\lambda x.\ x)\ (\lambda x.\ x)) = ((\lambda x.\ x), (\lambda x.\ x)) \tag{3.2}$$

indeed. We can thus again see that the evolutionary operators implicitly define a goal of the evolutionary process without any additional fitness. In this case, the goal points towards self-copying. Talking about information patterns Dawkins [12] also refers to that kind of behavior as self-replication.

Neural Network Soup. We construct a neural network soup $\mathcal{E}$ for neural networks of a given architecture with r weights. We randomly generate individuals from the search space $\mathcal{X} = \mathbb{R}^r$. We use the operators *self-train* (an instance of *live*) and the *attack* (an instance of *meet*) as defined in Section 3.1. In [31] we observe that the *self-train* operator has a stabilizing effect on the particles.[9] Initially, the *attack* operator can cause

[5] Fontana and Buss [22, 23] do not directly eliminate one of the particles involved in *meet* but simply add the result of the *meet* operator to the population and then perform random selection to reduce the population to its original size in between generations. Since the operands of *meet* are chosen fully at random as well, we expect no difference between these behaviors.

[6] Again, Fontana and Buss [22, 23] simply choose a finite horizon for the amount of steps of β-reduction and discard every particle beyond that for ease of computation.

[7] They cite [19] for the meaning of "hypercycle" here.

[8] This notion of fixpoint for the *meet* operator, i.e., effectively a fixpoint for λ-application, must not be confused with the whole theory of fixpoints of arbitrary functions in λ-calculus, which are fixpoints for β-reduction. The most famous such fixpoint for β-reduction is probably $(\lambda x.\ x\ x)\ (\lambda x.\ x\ x)$, which reduces to itself.

[9] Intuitively, that training performed here would clearly steer the networks towards learning the identity function *if* they would be trained with inputs across the input space. In our instance they are only trained with the one configuration of weights they just happen to have themselves. We find it curious that the operator *self-train* still has that effect.

great disturbances as it can completely rewrite the “attacked” network. To observe the emergence of stability, we need to relax Definition 16 a little to accommodate real-valued function representations such as neural networks:

Definition 17 (ε-Fixpoint [31]). *Given a neural network $\mathcal{N}$ with weights $\overline{\mathcal{N}} = \langle v_i \rangle_{0 \leq i < |\overline{\mathcal{N}}|}$. Let $\varepsilon \in \mathbb{R}$ be the error margin of the fixpoint property. Let $\mathcal{N}' = \mathcal{N} \lhd \mathcal{N}$ be the self-application of $\mathcal{N}$ resulting in weights $\overline{\mathcal{N}'} = \langle w_i \rangle_{0 \leq i < |\overline{\mathcal{N}'}|}$. We call $\mathcal{N}$ an ε-fixpoint iff for all i it holds that $|w_i - v_i| < \varepsilon$.*

Effectively, we will never be able to reproduce the exact same weights for a network since we are working in continuous space, but we can expect to get as close as we want. Our experiments in [31] further show that a soup as described above does indeed tend to produce ε-fixpoints. Perhaps due to the continuous nature of the weights, a vast amount of such ε-fixpoints can be found throughout the search space. We cannot (yet) rank them in any meaningful way as they all have the same complexity (because the network architecture is fixed). Due to an important observation made by Schoenholz et al. [70], the null network $\overline{\mathcal{N}} = \langle 0 \rangle_{0 \leq i < |\overline{\mathcal{N}}|}$ is much easier to find than other fixpoints starting from randomly initialized networks. However, the addition of *self-train* allows particles to snap into non-trivial fixpoints that may be closer to their initial position in the search space.
Figure 3.1 represents an attempt to visualize the behavior of an entire soup. From the initial population we see some particles follow a short smooth curve and then remain pretty constant for longer periods of times. These are typically ε-fixpoints reached after some generations of *self-train*. When particles perform larger jumps within the state space, they have been involved in an *attack* operation. We can see in Figure 3.1, which shows a typical run of our neural soup setup, that particles tend to cluster together after an attack; we clearly see some kind of goal area, towards where the particle soup is heading. However, note that *where* the soup will thicken is different between runs (and in some runs, there will not be any tendency to form clusters at all). We can thus conclude that the placement of the perceived goal area is not only implicitly given by the operators but also by dynamic properties of the soup, including its random initialization and the random choice of operator executions.
In the course of this chapter, we have thus not only seen that implicit goals may exist in evolutionary processes even when no explicit goals are specified, but we have also seen that these implicit goals may be self-adaptive in in every meaning of the word (including Definition 14).

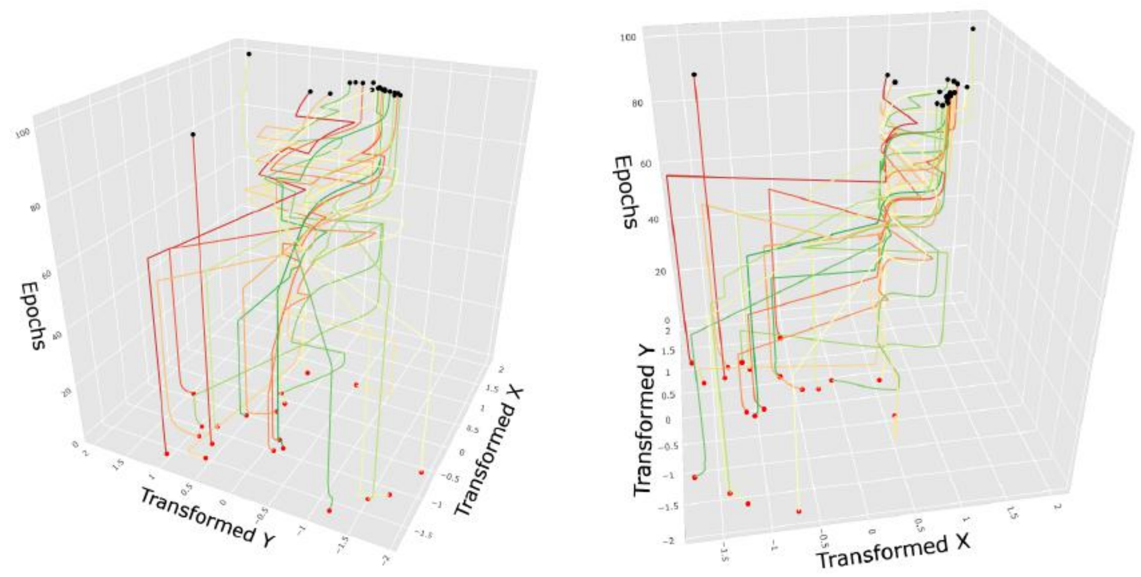


Figure 3.1.: Run of one soup consisting of 20 neural networks. The 20 neural networks $\mathcal{N} : \mathbb{R}^4 \to \mathbb{R}$ with two hidden layers with two cells each were initialized randomly and then evolved for 100 epochs. Per epoch, every network had a chance of 0.1 to *attack* another network and was subjected to 30 iterations of *self-train*. This setup allowed for emergent behavior of the network forming a cluster at a region of all non-zero fixpoints. The figure shows two perspectives on the same three-dimensional graph. The 20 weights in total per network were visualized in a two-dimensional space based on the transformed bases X and Y derived via principle component analysis (PCA). Image taken from [31].

4. The Ideal Fitness

Φύσις κρύπτεσθαι φιλεῖ.
Nature likes to remain hidden.

Heraclitus[1]

4.1. Diversity-Aware Fitness

In Chapter 3 we have seen that even without any explicitly given fitness function, individuals can show properties that resemble fitness. In this chapter we show that such implicit goals persist even when we give an explicit fitness function as well. As evolutionary processes with a given fitness we will examine evolutionary algorithms as described in Definition 6.
In simpler instances of evolutionary algorithms the overall target of the evolutionary process is directly passed on to the evolutionary process itself by the means of the fitness function, i.e., we use a fitness function $f(x, \_, \_) = t(x)$ given a target function t. And generally speaking, these instances work towards optimizing for that given target function t. So where is any room for additional implicit goals?
A vast body of literature discusses tweaks and variants of evolutionary algorithms.[2] In some way or the other, these always customize the evolutionary algorithm for a specific class of target functions, even when that class cannot be accurately described (and is rather implicitly given by the very nature of the modifications). This follows from the *No Free Lunch Theorem* [87]. We can conjecture that adjustments to an evolutionary algorithm's operators, search space, or other components improve the optimization's results iff the implicit goals they introduce match the explicit goal better. For example, if we transform the search space so that we can rule out a large amount of undesirable solutions for a given target function t, we may be able to improve the evolutionary algorithm's performance for target function t at the expense of deteriorating performance when we actually do want to find said solutions.
However, by using evolutionary algorithms in the first place, we are already committing to a certain kind of specialization over arbitrary search algorithms (for which No Free

[1]This quote by Heraclitus is found in Themistius's *Orationes* (Section 5, 69b) as part of a larger analysis.
[2]Since large parts of this thesis's bibliography already fall into that category, we refrain from citing any specific works here.

Lunch holds) and thus there are likely to be implicit goals that all evolutionary algorithms share. When using evolutionary algorithms, we assume that the fitness landscape (i.e., the projection $f\langle\mathcal{X}\rangle$ of the fitness or target function over the whole search space) has some sense of locality, i.e. there does exist a similarity relation $\sim$ so that for all $x, x' \in \mathcal{X}$ it often holds that $x \sim x' \implies f(x) \sim f(x')$.[3] If we cannot find such a relation $\sim$, evolutionary algorithms are probably the wrong tool for this fitness or target function.[4]

Exploration vs. Exploitation. The existence of locality allows for two major strategies when looking for new solutions: Exploration is the strategy of covering as much area of the search space as possible and looking for new solutions in areas where no individuals yet exist. Exploitation is the strategy of looking for the best possible solution that can be easily reached from known individuals, i.e., trying to not overlook optima in areas that already have been covered to some extent. Like all metaheuristic search algorithms, evolutionary algorithms would like to do both strategies to their full extent but due to resource constraints need to decide on which strategy to focus on, giving rise to the *exploration/exploitation dilemma*. Generally speaking, it is assumed that a "healthy" search process starts with a focus on explorative operations (random initialization, high mutation) and then gradually shifts towards more exploitative operations (high selection pressure, greedy local search) [10].
A lot of adjustments to evolutionary algorithms focus on finding the right balance between exploration and exploitation. We can implement some adjustments on a purely structural level, but we argue that we see their effect more clearly when we implement them as part of the fitness function. For example: If we want a more random set of survivors selected for the next generation (thus reducing the selection pressure), we might alter the selection function s^{sur} to draw a more random set or we might adjust the fitness function $f'(x, \mathcal{E}, r) =_{def} f(x, \mathcal{E}, r) \boxplus v(r)$ where $\boxplus$ is the clipped addition[5] on $[0; 1]$ and $v(r)$ might return a random value from $[-0.1; 0.1]$.

The Case for Diversity. There exist a variety of ways do define diversity within the context of an evolutionary algorithm; most of these ways are summed up by Squillero and Tonda [76]. Intuitively, diversity describes how spread out the population is within

[3] The usage of "often" in this formulation is meant to circumvent any probabilistic formulation like the more formal $\mathcal{P}(f(x) \sim f(x') \mid x \sim x') > \mathcal{P}(f(x) \not\sim f(x') \mid x \sim x')$. It is one of the strong points of stochastic optimization in general that assumptions on the problem need not hold strictly but only approximately.

[4] Prominent examples of problems that preclude themselves from evolutionary optimization might be hash functions, which are deliberately designed to violate said property of locality. We should expect an evolutionary algorithm trying to solve a hash function to actually perform worse than random search as it will waste time trying to build up schemas [79] that do not exist in that target function. Luckily, many practical optimization problems do have some sense of locality.

[5] We can define clipped addition analogously to clipped multiplication, for which we gave a full definition in Equation 3.1 (Section 3.2).

the search space. When we define a diversity metric on the population as a whole, we can then use it to steer the evolutionary process towards exploration or exploitation via, for example, adjusting mutation rates [83]. However, there also are measurements for the diversity of a single individual (against the backdrop of the rest of the population, of course). These again can be averaged and used on the whole population, but they can also used to affect the evolution of specific individuals, which is most easily done by building diversity into their fitness. Note that a diversity metric between two single individuals can act as similarity relation $\sim$ on individuals, which we discussed earlier in this section.

Definition 18 (Diversity-Aware Fitness). *Let $\mathcal{X}$ be a state space. Let $t : \mathcal{X} \to [0; 1]$ be the target function of an evolutionary process. Let $d : \mathcal{X} \times \mathfrak{E} \times \mathfrak{R} \to [0; 1]$ be a measurement for the diversity of an individual within a given evolutionary process (possibly based on some random effect). Let $\zeta \in [0; 1]$ be the diversity weight. A fitness function $f^+ : \mathcal{X} \times \mathfrak{E} \times \mathfrak{R} \to [0; 1]$ is called a* diversity-aware fitness function *iff it is of the form*

$$f^+(x, \mathcal{E}, r) = (1 - \zeta) \cdot t(x) + \zeta \cdot d(x, \mathcal{E}, r).$$

Of course, we could augment any arbitrary fitness function with diversity this way, but for simplicity we focus on only combining the objective target function with diversity measurements. It is most important to point out that any diversity-aware fitness f^+ is self-adaptive (cf. Definition 14) as the population changes during evolution and so does (potentially) every single individual's fitness.[6] Also note that vast amount of possibilities to define a diversity measurement d exists and the diversity weight ζ adds another hyperparameter to the algorithm; we discuss both of these issues in [28], where we compare various means of achieving diversity for a few benchmark problems.
When we use diversity-aware algorithms as we did in the studies in [26, 29, 28], we can observe the rather curious phenomenon we brought up in Chapter 1: Using an augmented (diversity-aware, e.g.) fitness f^+ instead of using the target function t directly as fitness we can actually end up with better results *with respect to* t [26, 29, 28, 35]. Effectively, adjusting our fitness function away from t ends up making us optimize for t better. In Section 4.2 we argue that this happens because the evolutionary algorithm approximates an implicitly defined inherent goal. For now, we rejoice that such relatively easy methods exist to improve our optimization process.

Manhattan Diversity. We would like to point out a seminal result by Wineberg and Oppacher [85]: "In this paper we have shown that all [diversity measures] are restatements or slight variants of the basic sum of the distances between all possible pairs of the elements in a system." Thus, we can with some confidence use a rather simple pairwise distance for d and still not miss out on much. Our experimental studies (mainly in [28],

[6]This also implies that we transform any target function t into a dynamic optimization problem at the level of the fitness function f^+. We will discuss some implications of that shortly.

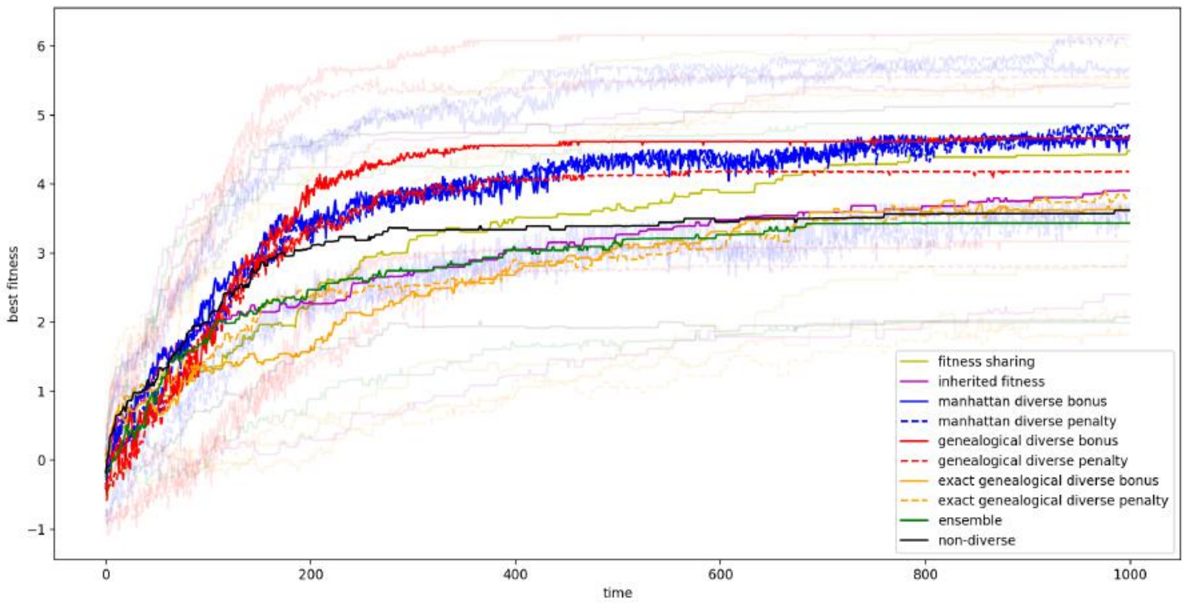


Figure 4.1.: Evaluation results for diversity-aware evolutionary algorithms using various diversity measurements. They solve the Pathfinding problem, which we introduced to specifically test for evolutionary algorithms that favor diversity. For each generation, we plot the current population's best objective value. Averaged over 20 independent runs. Semi-transparent lines show plus/minus one standard deviation. Image taken from [28].

cf. Figure 4.1) point into the same direction for using the pairwise Manhattan distance between individuals, i.e., $d(x, \mathcal{E}, r) = \sum_{x' \in X} \textsc{Manhattan}(x, x')$ where X is the latest population of $\mathcal{E}$.[7] We have further shown that we can approximate this measurement efficiently by choosing a random subset of the latest population for X in the previous formula [26, 29, 28].

Genealogical Diversity. Nonetheless, we still introduce a new method for measuring diversity in [26]: The main advantage of *genealogical diversity* is not in its results (as we discussed in the previous paragraph) but in fact that it does not require any metric to be defined on the search space $\mathcal{X}$. Instead, we augment individuals with a sequence of bits that is initialized at random and thereon subject to mutation and recombination but not the fitness calculation. A simple Hamming distance on that bit string is then a sufficient approximation for the distance between two individuals in the search space

[7]Please note that Wineberg and Oppacher [85] explicitly mention only the Euclidean distance for real-valued vectors. Due to the close similarity between Manhattan and Euclidean distance (compared to other diversity measures tested) and our applications in non-Euclidean space, we opt for the Manhattan distance here.

$\mathcal{X}$.[8] Of course, completely unrelated individuals might end up with the same bit string by chance, but that is enormously improbable for sufficiently large bitstrings. Instead, individuals with similar bit strings have probably had a common ancestor and thus probably tend to be more similar in all their properties. Figure 4.1 illustrates that despite all approximations involved genealogical distance can work as a diversity measurement quite like Manhattan distance.

Diversity and Dynamic Optimization. As we already mentioned briefly, incorporating a population-dependent diversity measurement into the fitness function turns the evolutionary process into a dynamic optimization process as the fitness $f(x, \mathcal{E}, r)$ will change over time even for a fixed $x \in \mathcal{X}$. What sounds like an additional complication at first may actually be a more natural way to describe evolutionary goals, as we shall explore in the following section. For the remainder of this section, we would like to briefly point to the reverse phenomenon: What does diversity do when we do face an inherently dynamic optimization problem, i.e., when the target function t does also depend on the current state of the evolutionary process $\mathcal{E}$.[9] In [29] we consider optimization problems where the target function changes halfway through optimization and we observe a phenomenon shown in Figure 4.2: For a simple optimization problem, diversity-aware evolutionary algorithms optimize for the best result a bit slower, but when the change in the target function occurs, they lose a lot less fitness as their populations are much less specialized and still maintain more alternative solutions. Especially for applications that depend on a certain robustness of the optimization process in the face of changing goals, maintaining diversity brings an additional benefit.

4.2. Productive Fitness

As we have seen in the previous section that using a different fitness $f \neq t$ might actually help to optimize the target function t, there is but one pressing follow-up question: If all we want to do is optimize for t, which fitness function should we then choose? Incidentally, we can trace the genealogical relations to give an inductive answer: At the very last generation, we should just use the target function t as fitness. The last population X_g contains all results we are going to get, so we might as well choose the best individual $x^* = \arg\max_{x \in X_g} t(x)$. For the second-to-last population, we should then assign a fitness that corresponds to the best result we are going to get in the last

[8]The idea is to approximate the genealogical relations between individuals just like biology traces the genealogy of species from matches in (mostly unused) genes in their DNA. This works even better when the considered genes have not been put under much selection pressure as they then retain the history of the accumulated mutations.

[9]Our definitions (mainly Definitions 2 and 3) do not allow for that, strictly speaking. We chose that form because we only discuss dynamic target functions in this short paragraph and want to spare the boilerplate for the rest of this thesis. We feel the necessary adjustment is rather trivial.

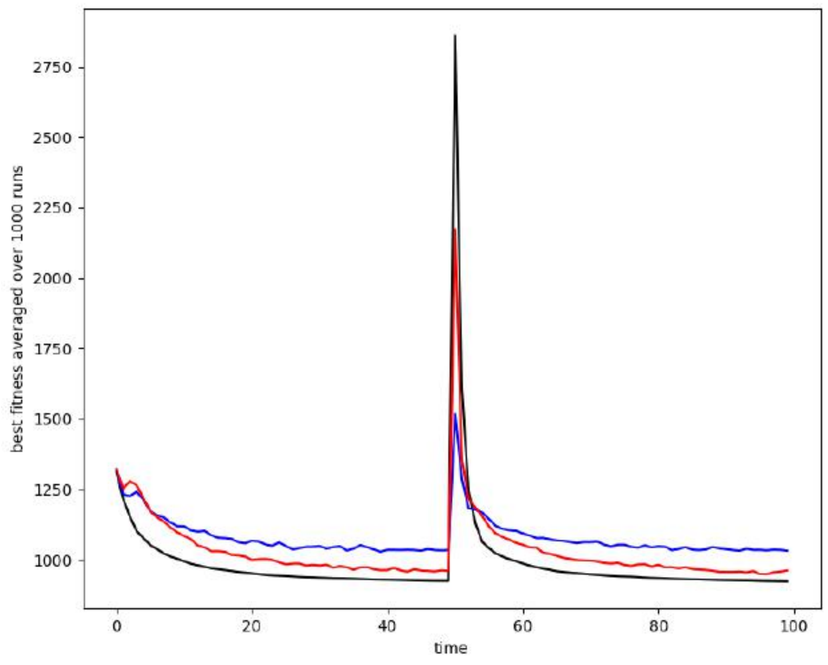


Figure 4.2.: Best (i.e., lowest-valued) fitness per current generation for a non-diversity-aware evolutionary algorithm (black) and diversity-aware evolutionary algorithms using Hamming (blue) und genealogical (red) diversity. They solve the Factory problem we introduced to test for phenomena of dynamic optimization. Results are averaged over 1000 runs. While the standard evolutionary algorithm (black) shows a steep spike at the time of the target function change (after 50 generations), the others (blue and red) manage to mitigate the negative amplitude to considerable extent. Image taken from [29].

generation. In a way, all we need to make sure in the second-to-last generation is that the ancestor(s) of the future best individual x^* survive. If we are just so optimistic, we might thus assign as fitness to each individual the best fitness its children will be able to contribute to the evolutionary process. Thus, for the second-to-last generation's population X_{g-1} we might use a fitness $f_{g-1}(x, \mathcal{E}_{g-1}, r) = \max_{x \in X'_g} t(x)$ where X'_g is the population that evolves from X_{g-1} (i.e., the population of $\mathcal{E}_{g-1}$) when the given individual x actually participates in that population.[10] We can then continue this chain of thinking until we reach the beginning of the evolution. Simply put, we want to value

[10]Of course, evolutionary processes are usually stochastic in nature so there usually will not be a single definite follow-up population "if x survives". Instead we will be facing a multitude of populations that may include x and may or may not have used various evolutionary operators on x. If we are optimistic, we might want to "max" over these possibilities to gain a fitness estimate; we might also choose another path and "avg" over all possible successor population, if we can approximate their probability distribution in any way.

the fitness of individuals throughout an evolutionary algorithm by the target value that the evolution will achieve because of them.
We call that fitness estimate with respect to future effects *productive fitness* (as it considers the individuals that will be produced in the future) and it can be formally defined as follows:

Definition 19 (Descendants [33, 35])**.** *Given an individual x in the population of generation i, $x \in X_i$, of an evolutionary process $\mathcal{E}$. All individuals $x' \in X_{i+1}$ so that x' resulted from x via a mutation operator, i.e., $x' = mut_r(x)$ for some $r \in \mathfrak{R}$, or a recombination operator with any other parent, i.e., there exists $y \in X_i$ so that $x' = rec_r(x, y)$ for some $r \in \mathfrak{R}$, are called* direct descendants *of x. Further given a series of populations $(X_i)_{0<i<g}$ we define the set of all descendants D_x as the transitive hull on all direct descendants of x.*

Definition 20 (Productive Fitness [33, 35])**.** *Given an individual x in the population of generation i, $x \in X_i$, of an evolutionary process $\mathcal{E}$. Let $D_x \subseteq \mathcal{X}$ be the set of all descendants from x. The* productive fitness after n generations *or* optimistic n-productive fitness ϕ^+_n *is the average achieved target value of x's descendants n generations later, written with $\omega = 0$ for maximizing and $\omega = 1$ for minimizing processes*

$$\phi^+_n(x) = \begin{cases} \text{avg}_{x' \in D_x \cap X_{i+n}} t(x') & \text{if } D_x \cap X_{i+n} \neq \emptyset \\ \omega & \text{otherwise.} \end{cases} \tag{4.1}$$

Definition 21 (Final Productive Fitness [33])**.** *Given an individual x in the population of generation i, $x \in X_i$, of an evolutionary process $\mathcal{E}$ with g generations in total. The final productive fitness of x is the fitness of its descendants in the final generation, i.e.,*

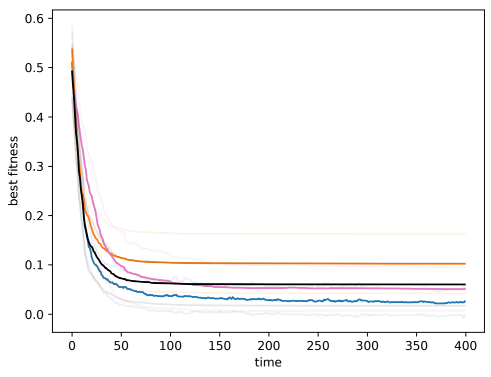
$$\phi^\dagger(x) = \phi^+_{g-i}(x). \tag{4.2}$$

In [33] we sketch a formal framework that allows us to argue that the final productive fitness for a given target function t is indeed the ideal fitness to use for the evolutionary algorithm. Intuitively. productive fitness can be understood as a different spin on the exploration/exploitation dilemma: Early in evolution, when exploration is favored, we allow for great deviations between the fitness of x and the target value of x, i.e., $|\phi^\dagger(x, \mathcal{E}, r) - t(x)| \gg 0$. Later in evolution, the fitness more and more approximates the target function until in the final generation $\phi^\dagger(x, \mathcal{E}, r) = t(x)$.
So if we know what fitness to go for, why do we still call it an exploration/exploitation *dilemma*? Actually computing the productive fitness involves an aggregation over all possible follow-up generations *for each generation* we need to look ahead. Obviously, the necessary amount of computation grows exponentially and is much too large for us to actually estimate productive fitness during a run of evolution. So far productive fitness is but an ideal target and a theoretical tool of analysis.
However, in [35] we also show a computable a posteriori approximation for productive fitness: After a complete run of an evolutionary algorithm (obviously using a different

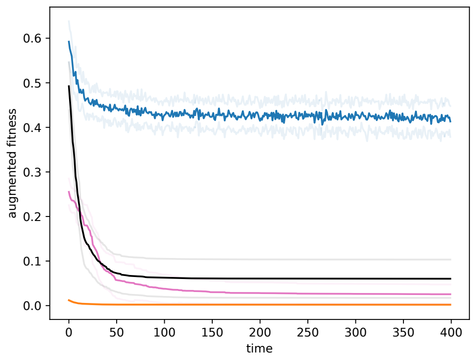
fitness function) has finished, we can go back and approximate the productive fitness we would have assigned to all individuals. This is still an approximation as in generation i we do not consider all possible future evolution paths but only the one future evolution path that did in fact happen, i.e., $X_{i+1}, ..., X_g$. However, since the realized evolution path is basically drawn at random from the set of all possible evolution paths, we argue that it is viable to approximate the expectation value of all possible paths via the average of the realized ones, at least when performing lots of runs of evolution. Nonetheless, using an approximation of the future via an a posteriori look at what was to come is the only (computationally) viable option anyway.

Fortunately, the results of said a posterior approximation for various evolutionary algorithms are quite intriguing: When we compare evolutionary algorithms with different fitness functions, we find that those algorithms have a better overall performance whose fitness throughout the evolution approximates the final productive fitness better. This provides some empirical evidence that adjustments to the fitness functions might be helpful exactly when they bring the fitness values closer to the final productive fitness $\phi^\dagger$. Figure 4.3 shows some of these results: When we instantiate evolutionary processes to optimize the standard Schwefel benchmark function [15], we can see that Manhattan-distance-based diversity-aware evolution has a slight edge with respect to to the target function t (Fig. 4.3a). We can also confirm that the fitness f used during evolution deviates from t quite a bit for the diversity-aware evolution (Fig. 4.3b). When we compute our a posteriori approximation of final productive fitness $\phi^\dagger$, we can see the eventual advantage shine through even in the beginning of evolution (Fig. 4.3c). Now we compute the difference between the approximated final productive fitness $\phi^\dagger$ and the fitness function f that was actually used in evolution and see a rather constant difference value throughout evolution for the diversity-aware case (Fig. 4.3d), implying that the diversity-aware evolution matches the final productive fitness much more closely than the others, including the naïve approach with $f(x, \_, \_) = t(x)$.

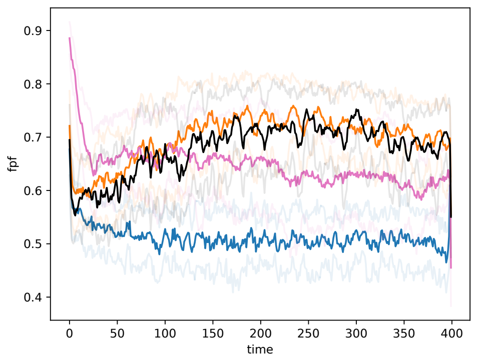
We should note that final productive fitness is again inherently a dynamic metric, thus transforming any target function t into a dynamic optimization problem. We can see that final productive fitness includes the meta-optimization problem of when to use which parameters of the search (when to focus on exploration or exploitation, e.g.) into the fitness function. This shows us that evolutionary algorithms behave in a self-adaptive way even when they are solving a static optimization problem.



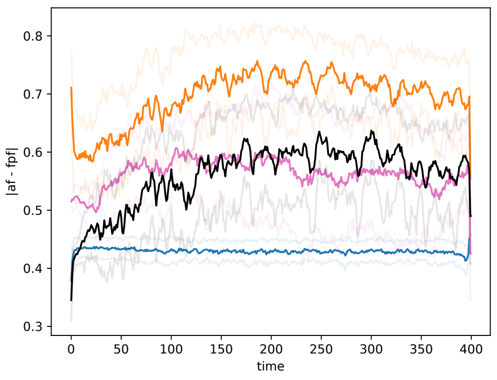
(a) Best objective fitness t.



(b) Adjusted fitness of the objectively best (according to t) individual per generation.



(c) Best final productive fitness $\phi^\dagger$.



(d) Average per-individual difference between adjusted fitness f' and final productive fitness $\phi^\dagger$.

Figure 4.3.: Evolution for the classic Schwefel problem. Standard evolutionary process using $f(x, \mathcal{E}, r) = t(x)$ shown in black, diversity-aware evolutionary process using a diversity-aware fitness $f(x, \mathcal{E}, r) = 0.5 \cdot t(x) + 0.5 \cdot d(x, X)$ with pairwise Manhattan distance d shown in blue. Inherited fitness (purple) and fitness sharing (orange) shown for comparison. Results averaged over 20 independent runs, standard deviation is shown in transparent lines. Images taken from [35].

5. Co-Evolutionary Adaptation of Fitness

Εἰδέναι δὲ χρὴ τὸν πόλεμον ἐόντα ξυνόν, καὶ δίκην ἔριν, καὶ γινόμενα πάντα κατ′ ἔριν καὶ χρεών.
One must know that war is a common thing and that the law is conflict and that everything evolves through conflict necessarily.

Heraclitus[1]

5.1. Scenario Co-Evolution for Reinforcement Learning

In the previous chapter we have discussed that fitness functions for evolutionary algorithms should ideally approximate the final productive fitness. We gave a first example where diversity-aware fitness functions provide such an approximation to a relatively good extent. However, we have no claim that diversity-aware fitness is the best feasible approximation of productive fitness or that it is even good for all settings of evolutionary processes. So we end up in a situation where we are looking for an efficient and close approximation of an ideal goal. Sounds like an optimization problem!

Meta-Optimization and Memetic Algorithms. There exist a lot of approaches that consist of adapting parts of the evolutionary algorithm while the algorithm itself is running and solving for t. Simple approaches might automatically determine the mutation rate α_{mut} by simply adding it to the genome, subjecting it to mutation and recombination but exempting it from the fitness evaluation [59]. This helps us treat the exploration/exploitation dilemma automatically (to some extent) and reduce the number of hyperparameters we need to account for manually. Because we are optimizing the hyperparameter while we are optimizing the target function, we call the former *meta-optimization.* More complex approaches exist like *memetic algorithms* [61]: Although

[1]This quote by Heraclitus survived via a mention in Celsus's "Λόγος Ἀληθής" ("The True Word"), which in turn only survived via the citations made by Origen of Alexandria in his answer "Κατὰ Κέλσου" ("Against Celsus").

somewhat confusingly named[2], they just perform meta-optimization on more than mere reals but are able to automatically adjust the whole behavior of the operators or larger parts of the evolutionary algorithm. To perform these adjustments, they often construct a second evolutionary process. This co-evolutionary process is rather loosely coupled with the main evolutionary process optimizing individuals for t but of course can steer its way throughout optimization.

Definition 22 (Co-Evolutionary Process)**.** *Let $\mathcal{X}, \mathcal{Y}$ be a state spaces. Let $G : \mathfrak{P}(\mathcal{X} \times \mathcal{Y}) \to \mathfrak{P}(\mathfrak{P}(\mathcal{X} \times \mathcal{Y}))$ be the* joint evolutionary function*. Let $E : \mathfrak{P}(\mathcal{X}) \to \mathfrak{P}(\mathfrak{P}(\mathcal{X}))$ be an and $F : \mathfrak{P}(\mathcal{Y}) \to \mathfrak{P}(\mathfrak{P}(\mathcal{Y}))$ be evolutionary functions so that $G((X, Y)) = (E(X), F(Y))$. Let $v : \mathcal{X} \times \mathcal{Y} \to [0; 1]$ be the* joint target function*. Let $s_1^{\mathcal{E}}$ and $s_1^{\mathcal{F}}$ be suitable selection functions. Let $t : \mathcal{X} \to [0; 1]$ be a target function of the form $t(x) = v(x, s(Y, \mathcal{F}, r))$ for given $Y \subseteq \mathcal{Y}, \mathcal{F} \in \mathfrak{E}, r \in \mathfrak{R}$ and $u : \mathcal{Y} \to [0; 1]$ be a target function of the form $u(y) = h(v(s(X, \mathcal{E}, r), y))$ with $h : [0; 1] \to [0; 1]$ for given $X \subseteq \mathcal{X}, \mathcal{E} \in \mathfrak{E}, r \in \mathfrak{R}$. An evolutionary process $\mathcal{G} = (\mathcal{X} \times \mathcal{Y}, G, v, \langle Z_i \rangle_{i \leq g})$ is called a* co-evolutionary process *iff $\mathcal{E} = (X, E, t, \langle X_i \rangle_{i \leq g})$ and $\mathcal{F} = (Y, F, u, \langle Y_i \rangle_{i \leq g})$ are evolutionary processes.*

With this rather lengthy definition we describe a setup where two evolutionary processes $\mathcal{E}$ and $\mathcal{F}$ can (for each evaluation of their respective target functions t and u) select an individual from the other evolutionary process to call a joint target function v. Together they form a co-evolutionary process $\mathcal{G}$. Note that u calls a transformative function h: For $h(w) = w$ we call $\mathcal{G}$ cooperative co-evolutionary, for $h(w) = 1 - w$ we call $\mathcal{G}$ competitive co-evolutionary. All co-evolutionary processes have self-adaptive fitness as the value of the target functions t and u can change depending solely on the other evolutionary process.[3]
As discussed, various instances of co-evolutionary processes for two (or more) evolutionary algorithms exist. However, the framework seems more general and in [32, 36, 37] we have shown that it can also be applied to more hybrid cases: We used a reinforcement learning agent that (competitively) co-evolved with an evolutionary algorithm.

Reinforcement Learning. Issues like the exploration/exploitation dilemma supersede the field of evolutionary algorithms; in this thesis we consider reinforcement learning as

[2]As we mentioned briefly in the Introduction (Chapter 1), the term “meme” was originally conceived by Dawkins [12] as an example for an existing replicator that is not RNA/DNA-based. He thus considered a meme as the unit of cultural evolution and gave rise to a small field of research on that idea [6, 14] as well as the commonly used name for funny pictures and sayings on the internet, “internet memes” [4]. Memetic algorithms are named as they are to show that they employ a second replicator, i.e., a second intertwined evolutionary process, just like humans are subject to both biological and cultural evolution. However, they have no connection to any cultural aspects or imitation learning that are substantial parts of the definition of memes.

[3]This implies that both $\mathcal{E}$ and $\mathcal{F}$ have adaptive fitness with respect to their environment consisting of the other process respectively. Put together, the co-evolutionary process $\mathcal{G}$ has the same adaptiveness with respect to what is now its own components and thus features self-adaptive fitness.

a central representative of the current research in artificial intelligence. For a more in-depth introduction to reinforcement learning we refer to the respective section in [36] or directly to standard literature [77]. In [37] we also provide a formal integration of stochastic gradient and other methods commonly used in state-of-the-art reinforcement learning into our notion of adaptive processes. Within the formal framework we described until now, we can easily write a process of reinforcement learning as an evolutionary process[4] with a population size of $|X| = 1$ over a search space $\mathcal{X} = \Pi$ where each $\pi \in \Pi$ is called a policy and is some representation of a function $\pi : \mathcal{O} \to \mathcal{A}$ from a space of observations $\mathcal{O}$ to actions $\mathcal{A}$. Usually, this representation is given as weights of a neural network (that is used within a fixed surrounding function). The problem for a reinforcement learning agent is given as an accumulated reward function $\mathcal{R} : \Pi \times \mathcal{S} \to [0;1]$ that (at some level of abstraction[5]) assigns a total reward $R \in [0;1]$ to each policy $\pi \in \Pi$ and each initial configuration $s \in \mathcal{S}$ of the agent's environment. As we assume that all of these initial configurations are equally likely for the agent to find itself in, the target function of a reinforcement learning process commonly looks like this:

$$t(\pi) = \frac{1}{|T|} \cdot \sum_{s \in T} \mathcal{R}(\pi, s) \tag{5.1}$$

Here, $T \subseteq \mathcal{S}$ is the test set also called a set of *scenarios*, i.e., a selection of initial configurations on which each $\pi \in \Pi = \mathcal{X}$ is evaluated, since we usually cannot hope to test all of $\mathcal{S}$ exhaustively. The evolutionary function E would then usually compute a weight update for the policy-encoding neural network based on an aggregation of rewards (that might or might not match t).[6]

Scenario Co-Evolution as Auto-Curriculum Learning. The choice of the hyperparameter T for the set of test scenarios within the target function t has naturally been studied in literature. Common solutions are to just sample a feasibly large data set $T \subseteq \mathcal{S}$ on demand or to set aside such a (non-continuous) part of the search space before training begins. However, different scenarios might have a different difficulty to solve. In our studies [36, 37] we noticed a phenomenon we called the "exam effect": An

[4]Curiously, in what is basically the first description of the idea of artificial intelligence, Turing [81] already likens the process of machine learning via rewards and punishments to natural evolution.

[5]For the macro-level perspective we assume here, we omit how actions are mapped to total rewards and simply assume all of this is hidden within a function $\mathcal{R}$. Within the field of reinforcement learning, we usually assign reward based on single state-action pairs and and often even omit any computation of the accumulated reward $\mathcal{R}$ per policy and run.

[6]Note that there again might be a distinction between the reward function used within t that objectively measures what *we* want a reinforcement learning agent to do and the reward function that we tell *the agent* to use for its updates; this works analogously to target function t and fitness function f in evolutionary algorithms. The task of finding the right reward function for the agent from our goals is known as *reward engineering* and extensively studied in the field of reinforcement learning [86, 41]. We will make that connection once more in Section 7.3.

agent that solves all the hard scenarios usually also solves all the easy ones. We can compare this effect to how an exam at school or university does not need to include the easy questions. In another parallel to human learning, increasing the difficulty throughout the learning process (i.e., starting with easy scenarios and only then moving on to hard ones) is known to improve overall performance. This technique, called *curriculum learning*, is well-established in reinforcement learning [5, 57].

These observations give us a rough idea on what constraints there might exist on choosing an ideal scenario set T for our target function t. As it has been commonplace throughout this thesis, when faced with optimization problems, we opt for an evolutionary algorithm: We construct a competitive co-evolutionary process (cf. Definition 22) where a reinforcement learning agent tries to learn a policy π to achieve the best score $t(\pi)$ while an evolutionary algorithm is used to optimize for the test set T against which the reinforcement learning agent achieves the worst score $t(\pi)$.[7] We call that process *scenario co-evolution (SCoE)*; Figure 5.1 illustrates how it works. Our results in [36] show that scenario co-evolution gives rise to both better and more robust policies and additionally generates a set of hard test scenarios that can even be used independently of the originally trained agent. Most impressive, perhaps, is that even though we put in all this effort of executing the co-evolutionary process and running a whole additional optimization process, the score per wall clock time ratio is slightly better for scenario co-evolution compared to standard reinforcement learning (in our experimental setup). Frankly speaking, the effort pays off. Figure 5.2 shows the results from our experimental runs.

From a reinforcement learning perspective, scenario co-evolution falls into the category of *auto-curriculum* learning: The set of scenarios that always increases in difficulty provides a curriculum for the reinforcement learning agent, but it can only ever become more difficult in relation to the agent's performance. For an agent that cannot solve any scenarios, all scenarios look equally difficult. Thus, the agent itself defines the path of scenarios it is likely to experience during its training, forging its own curriculum. From an evolutionary algorithms perspective, scenario co-evolution is yet another instance of a self-adaptive fitness function: The same scenario that has once been evaluated to be tremendously difficult may cease to be so as the agent learns to solve it. Indeed the very fact that the scenario was difficult and thus included in many generations of scenario sets T may have prompted the agent to learn how to solve it. Thus, at least until the objectively worst scenarios for the objectively best agent are found, any fitness evaluation immediately results in a process of self-adaption (up to some probability), thus making self-adaptive fitness the central piece that holds this co-evolutionary process together.

[7]The naïve solution to find an optimal test set T might be to construct an evolutionary algorithm on a search space of all possible test *sets*, i.e., $\mathcal{X} = \mathfrak{P}(\mathcal{S})$. However, this search space is exceedingly large and the dependences between the singular test scenarios within T are very weak, especially in the beginning of the learning process. Thus, we choose a "short-circuited" variant where the search space is the space of single test scenarios, i.e., $\mathcal{X} = \mathcal{S}$, and we use the whole population to construct the test set, i.e., $T = X_i$ for every new generation i.

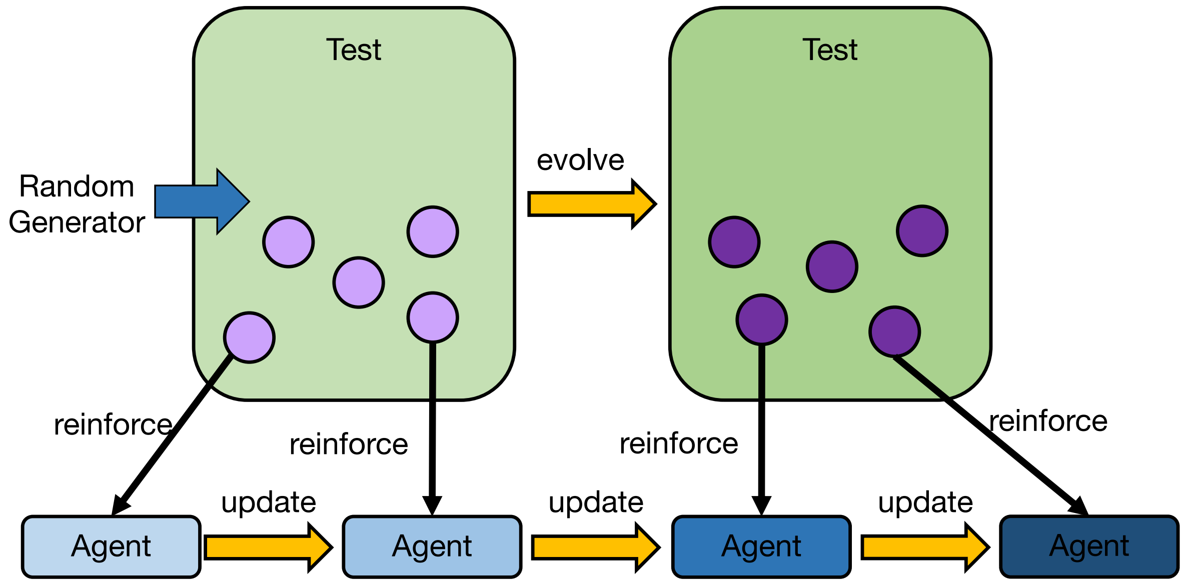


Figure 5.1.: Schematic representation of a SCoE process. A population of test scenarios is first generated at random and then improved via evolution. Between evolutions, the test scenario population is fully utilized as training data for the reinforcement learning agent, which causes the agent to improve in parallel to the test scenario population. Image taken from [36].

5.2. Organizational Patterns for Software Development

We introduced scenario co-evolution in the previous section as an interaction pattern between a reinforcement learning agent and an evolutionary algorithm where both contribute to a joint target function: The former provides a certain kind of behavior (i.e., a policy π) and the latter provides a set of test instances for that behavior (i.e., the set of test scenarios T). That kind of interface, however, is applicable way beyond just reinforcement learning agents and evolutionary algorithms. And as our formal definitions for (co-)evolutionary processes (cf. Definitions 3 and 22) are rather general, it is easy to see how we can "plug in" various kinds of evolution here.

One of those is software evolution, which we mentioned in the Introduction (Chapter 1) as rather curiously named the same way as natural evolution. As of now, the behaviors of software are still largely determined by human programmers and so are the test suites for them. In [32] and later in [37] we explain scenario co-evolution for the concrete example used in the previous section, but we expand its scope to encompass software engineering processes in general, even when the evolved artifacts are not machine-generated. From that perspective, the notion of "scenarios" works as a catch-all for artifacts that impose

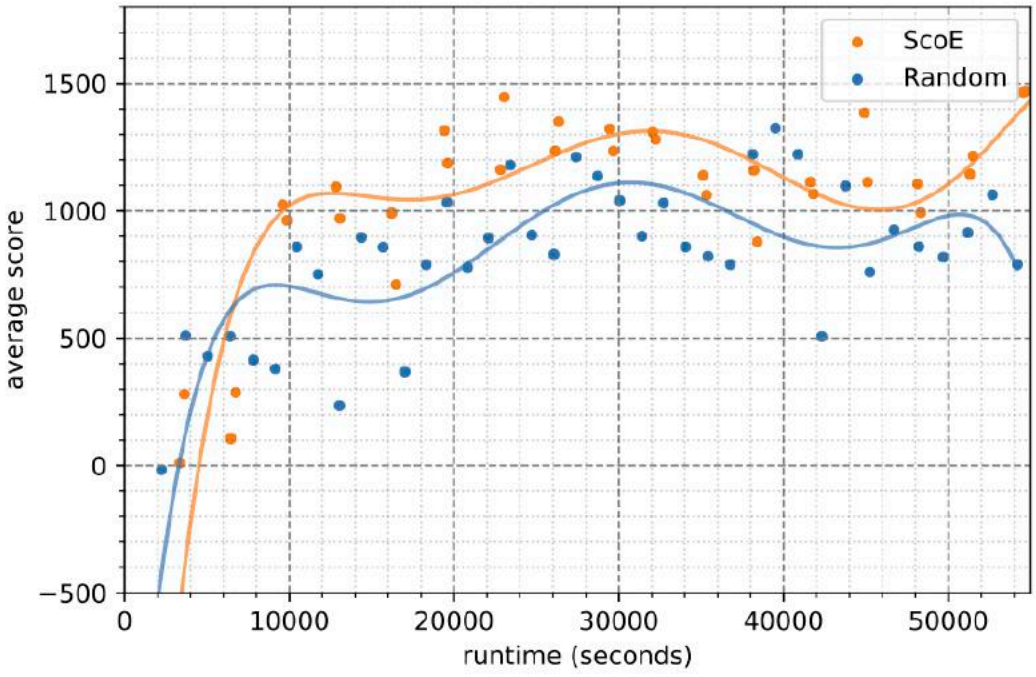


Figure 5.2.: Scores achieved by a SCoE and standard reinforcement learning (against test scenarios chosen randomly) during training for ≈ 50000 seconds of runtime. Scores are averages of running the current agent against 1000 randomly generated test scenarios. The plot shows single runs with an added trend line. Over the same amount of training time, SCoE generally achieves slightly higher average scores. Image taken from [36].

some kind of constraints, requirements or optimization targets on the software product. A living specification is maintained by keeping a large pool of such scenarios, which we treat as a competitively co-evolving adversary of the software product. What makes this perspective especially interesting is that it allows a seamless integration between contributions made by human developers and contributions discovered by autonomous evolutionary processes.[8] For such software systems, we identify three patterns to effectively steer the development process (of course there are many more to be discovered):

Criticality Focus. Facing a large (and possibly ever-growing) pool of scenarios, running a full test evaluation can become increasingly costly. We can alleviate the problem by constantly rating test scenarios for difficulty and prioritizing on the currently most difficult ones (similarly to curriculum learning for the reinforcement learning agent).

Adaptation Cool-Down. While traditional software engineering usually starts with a basic use case and adds features onto it, for software systems involving evolutionary processes, we often observe a reverse pattern: The system starts with a lot of freedom in its behavior (because of randomly initialized behavioral representations, e.g.) and as it co-evolves with the scenarios, its degrees of freedom get reduced further and further. For various stages of the product life-cycle ("in development" vs. "deployed on-site"), various degrees of behavioral freedom may be deemed appropriate.

Eternal Deployment. For constantly evolving systems, the border between development phases and deployment phases vanishes. Nierstrasz [63, 62] described large and complex software systems (the internet, e.g.) as *eternal systems* in that they are not only ever-changing but also ever-working: There is no discrete point in time to take them offline and fix them or update them. For systems employing evolutionary processes this means that we need to maintain not only a representation of the currently best configuration but also of how we got there.

Patterns like these help developers to understand how an effective evolution process should be designed, just like traditional software development patterns [39, 9] provide a richer language to understand software design. We should expect that the increasing amounts of automated processes, optimization and artificial intelligence that go into software development will eventually change the discipline of software engineering.[9] However, we feel like patterns that involve both deliberate and natural design at the

[8]We argue that achieving equal interfaces for human and machine contributions to software is an important target for AI-aided software design. We sketched first approaches based on a teacher-student interface in [46, 47], which of course builds upon Turing's description of a teacher-student-based evolutionary process [81].

[9]We argue that indeed software *engineering* may cease to be the metaphor used for making software and have occasionally suggested *software gardening* as a replacement term.

Tier	Description
0	**physical necessity:** laws of nature
1	**machine-environment interface:** circuit-level control, sensor/actuator hardware, computational capabilities
2	**immediate reaction:** watchdogs, fixed behavioral rule-sets, expert systems
3	**planned reaction:** reward functions for online planners and self-awareness
4	**inherent coordination:** co-evolution, adversarial learning, multi-agent coordination

Table 5.1.: Overview of different levels of control to be used in the information flow of a cyber-physical system. Tiers 0–3 taken from [27].

same time still seem a bit weird as practical guidelines. We think that future work will improve upon the recommendations for practical use.
We suggested some more concrete patterns for software design for self-adaptive systems already in [27]. Here we use the example of software-intensive systems that also interact heavily with the physical world; these are often called *cyber-physical systems* [7, 8]. We immediately recognize that interaction with the physical world more easily allows for failures of self-adaptation mechanisms to become really dangerous and thus that any state-of-the-art evolutionary process (ranging from evolutionary algorithms and planners to full-fledged reinforcement learning agents, e.g.) needs to be heavily checked and controlled for practical use. We organize possible means to check and control the results of evolutionary processes into various tiers as shown in Table 5.1. Most notably, the original depiction in [27] lacks tier 4 of Table 5.1 as originally it seemed a bit paradoxical to control an evolutionary process running wild using another evolutionary process that might just as well run wild. However, in the course of this thesis we have shown that there can be trust in higher-level principles and that even seemingly free self-adaption is governed by laws, even when they are not explicitly stated in software. Our results show us among lots of other things that co-evolution (when set up the right way) usually works towards the robustness of both processes by evolving more effective fitness evaluations instead of settling on laxer ones.

6. Applications in Natural Computing

Ἁρμονίη ἀφανής φανερῆς κρείττων.
Concealed harmony is stronger than evident harmony.

Heraclitus[1]

6.1. Variants on Evolutionary Algorithms

We now discuss a few results that have not been at the core of this thesis but nonetheless do concern evolutionary processes with self-adaptive fitnesses. We start off with two further studies on different evolutionary algorithms.

Surrogates. In many instances of evolutionary algorithms, evaluating the fitness function is the computationally most expensive operation of the process. For such instances, we can train a surrogate model of the fitness function; surrogate models are usually much simpler than the original fitness function and thus much cheaper to evaluate. But they can be used instead of evaluating the real fitness function and thus save time. Of course, we must still eventually resort to the true fitness function to be sure about the results of our evolutionary algorithm, but every evaluation of a costly fitness function that we can spare frees computation time for other operations. When we then consider the surrogate-based fitness function (that is used for as many fitness evaluations as possible), it is clear to see how it is self-adaptive: As the evolutionary process continues and we end up using the true fitness from time to time, we make double use of that true evaluation to further train our surrogate model. Thus, even without any change in the true fitness, the fitness suggested by the surrogate model will change on its own during evolution, hopefully getting closer and closer to the true fitness. Note that this also means that we again changed a static into a dynamic optimization problem.
In [25] we used a surrogate-assisted evolutionary algorithm not for the original purpose of simply saving computation time but in an attempt to construct entirely new applications from the surrogate model. We propose to use surrogate models for recommendation

[1]This quote of Heraclitean philosophy is once again recorded by Hippolytus (*Philosophumena*, Book IX, Chapter 5).

systems. In a recommendation system, we assume we have a search space $\mathcal{X}$ consisting of various items (music, movies, products in an online store, e.g.) and we want to pick the one which the user is going to like best. We assume we can measure how much a user likes a certain item only after we have chosen it.[2] However, once we have presented an item, it is removed from the search space $\mathcal{X}$ as we will never recommend the same item twice; this is a major difference to most evolutionary algorithms, of course, where a fitness evaluation does not alter the population or the search space. Now, we can still build a surrogate model based on the true-evaluated and thus removed items and use that surrogate model to evaluate other not-yet-chosen items, from which we can then choose the most promising one, true-evaluate it and thus continue the cycle. In [25] we show some early results where user interaction is replaced by standard benchmarking objective functions in order to test the algorithmic properties of the approach, showing that at least in principle such an approach can work.

Phases and Penguins. In [68] we consider an evolutionary algorithm to optimize a *test suite* for a specific self-organizing system (controlling an adaptive production cell as it might exist in a smart factory). Such a test suite is a set of *test sequences*, each of which is a series of *test cases*, which are performed in a specific order. As the system is self-organizing, the order in which the test cases are executed may effect the result greatly. Thus, assuming that $\mathcal{C}$ is the space of single test cases, our search space $\mathcal{X} = \mathfrak{P}(\mathcal{C}^*)$ is the space of all sets of sequences in $\mathcal{C}$, which can be quite cumbersome and expensive to navigate. A test suite $x \in \mathcal{X}$ is evaluated using a given metric $t(x)$, which counts the amount of errors (from a pre-defined set of possible misbehaved actions by the system) that become apparent when running all of x.[3] In order to handle the complex structure of the search space $\mathcal{X}$, we propose two extensions to standard evolutionary algorithms: The *phases extension* again works by turning an originally static optimization problem into a dynamic one. But this time we do so by dynamically adjusting the search space $\mathcal{X}$: We start the evolution on a simplified search space $\mathcal{X}_1 = \mathcal{C}^*$, which only contains test suites with a single test sequence. After a few generations we then expand the search space to $\mathcal{X}_2 = \mathcal{C}^* \times \mathcal{C}^*$ containing all test suites with two test sequences. When doing so, we augment all individuals present in the current population by adding a randomly generated second test sequence to them. We continue this process until we arrive at the pre-set maximum size for our test suites (and then continue evolution normally for a few generations).[4]
Still, we encountered some difficulty implementing the mutation and recombination op-

[2]For example, when we recommend music, we might measure if the user actually finishes listening to the song. We might also assume that we can use affective computing methods to measure *how well* the user liked a specific item.

[3]This method is called *mutant testing*; however, the usage of mutant here is different from our evolutionary operator. See [68] for the thorough explanation of our approach.

[4]In this case, a larger test suite is always at least as good as a smaller subset, i.e., $x' \subseteq x \implies t(x') \leq t(x)$ for all $x, x' \in \mathcal{X}$.

erators for this evolutionary algorithm: Not all test cases in $\mathcal{C}$ can follow a previous test case $c \in \mathcal{C}$ to form a valid test sequence, i.e., our search space is really $\mathcal{X} = \mathfrak{P}(\mathcal{Q})$ for some test sequence space $\mathcal{Q} \subseteq \mathcal{C}^*$. This means that the mutation operator may need to actually alter an individual to a large extent (when it wants to change a test case early within a test sequence and needs to re-generate the whole follow-up sequence) and that recombination can hardly combine two different test sequences from two parents (as the test sequence may start out differently and then not be compatible for crossover, e.g.). However, the domain does give us a similarity metric on test sequences $S : \mathcal{Q} \times \mathcal{Q} \to \mathbb{R}$ and we can use it to define the *penguin extension* to mutation and recombination.[5] For penguin mutation, we choose a random test case within a test sequence and alter it. Then, we fill the follow-up test cases of that sequence with the valid follow-up sequence that is closest to the original (and now probably no longer valid) follow-up sequence with respect to S. For penguin recombination, we choose a random crossover point within two test sequences (one from each parent), cut off the original sequence after the crossover point and re-fill it with the valid follow-up sequence that is closest (with respect to S) to the cut-off follow-up sequence of the other parent's chosen test sequence. Using these operators, we can apply intuitive mutation and recombination to data types with high internal dependencies (like paths in graphs) as long as we can construct a similarity function.

6.2. Quantum Computing

Natural computing is the field of research on computational representation of processes observed in nature. Basically all of the methods we describe in this thesis fit that description: Artificial chemistry systems mimic processes from chemistry, obviously. Neural networks are built after very simplified models of the workings of biological neurons. Reinforcement learning was first described in behavioral psychology. The workings of natural evolution are put into computer code in the form of evolutionary algorithms in various forms, even allowing for more intricate phenomena like co-evolution to be worked into the model. Of course, the field of natural computing expands beyond the examples we can show in this thesis. Still, we use this section to take a look at another computing technique that originates from a different science: Quantum-inspired algorithms draw inspiration from the behavior of particles at the quantum level and use the rules that govern them to solve typical computational tasks, like optimization problems for example [60, 42, 44].
However, *quantum computing* is the sub-field of natural computing that promises the most use of doing things "the quantum way": Here we use nature itself to execute the

[5] The extension are named for the following metaphor: Imagine we try to recombine a typical bird like a dove and a typical fish like a tuna. They are fundamentally incompatible, but we might take the bird and change what we can to make it as much like a fish as possible. The result is a penguin, a species that definitely is a bird but shares features from the recombination partner, i.e., the fish.

quantum mechanics part, more specifically we build a machine called a quantum computer to do that. On it, we can run specifically designed quantum algorithms that make use of the additional[6] computational capabilities of a quantum computer. Note that similar approaches exist within natural computing in the form of DNA computing, using real-world processes in the field of biology to solve computationally hard problems [64].

Quantum Annealing. In our work, we focus on the quantum annealing algorithm, an optimization algorithm which can be and practically is implemented in specialized hardware. Like most optimization processes it uses a target function t to optimize for elements of a search space $\mathcal{X}$. However, thanks to the quantum effects it can use, it does not need to maintain a population of possible solutions (in contrast to evolutionary algorithms) while still being able to search in multiple directions at the same time (in contrast to simulated annealing [54]). We provide a slightly more detailed introduction in [38] and would like to refer to [51, 50, 58] for a more in-depth analysis. What we would like to point out in the context of this thesis is the observations on inherent properties of the used goal functions in the context of quantum annealing: In the study in [38] we used quantum annealing to solve the canonical NP-complete problem 3SAT, translating the decision problem into an optimization problem along the way. We could observe that certain solutions are inherently preferred as outputs, even when they yield the same target value in theory. Effectively, inherent goals are present in quantum annealing as well, influenced by the solution encoding and/or the quantum annealing hardware.[7] To overcome current hardware limitations we also explored an approach to solve problems in the typical encoding used for quantum annealing without actually using quantum annealing. For this purpose, we implemented neural networks to emulate the behavior of quantum annealing with inconclusive but initially promising results [30]. This shows that underlying principles of optimization may span beyond the realm of evolution, sketching a more fundamental concept of computation based on the laws of nature.

6.3. Monte-Carlo Tree Search

Monte-Carlo Tree Search (MCTS) is an algorithm that mainly searches for policies of actions for Markov Decision Processes (MDPs). In that regard it is very similar to reinforcement learning (RL), which we briefly introduced in Section 5.1.[8] While RL

[6]The question if there is a quantum algorithm that (i) can be practically implemented in real hardware and (ii) runs better on a quantum computer than on any classical computer is called *quantum advantage* or *quantum supremacy*. We skip a discussion on nuances of meaning between these terms. Recent research points into the direction of the first such algorithms [1], but their practicality is still being discussed in the community [65].

[7]We go on to analyze certain aspects of the robustness of the returned solutions as impacted by classical solvers vs. quantum annealing in [69] and probably future research.

[8]Some may argue that MCTS could in fact be regarded as an *instance* of reinforcement learning. For this thesis, we discern the two, mainly for historical reasons [84].

usually encodes behavior via a value function (in the form of a table or neural network) or a policy network, MCTS builds up a decision tree following the action choices given by the MDP. Especially for discrete games, the combination of RL and MCTS has been shown to be extremely successful in recent years [71, 72]. We performed experiments on a simple grid-world domain with multiple rooms and on the game Tetris [34].
Both domains have repetitive structure that is evident for the human observer: Navigating within a single room always works in a similar way and so does controlling one tetromino[9] until it comes to rest. However, the actions of the domain are more elementary and thus do not reflect that repetition. In the grid-world domain, the target function only rewards successfully playing the whole game, while the target function for Tetris gives a reward after each single tetromino. We then construct a variant of MCTS based on subgoals: Given a subgoal predicate $g : \mathcal{S} \to \mathbb{B}$ on the observable configurations, we divide the planning process into (i) planning which subgoal we want to reach and (ii) planning how we get from one subgoal configuration to the other. This approach has rather little effect on the grid-world domain but shows tremendous improvement on the Tetris domain, where the chosen subgoal predicate matches the states where the target function yields a reward. For an in-depth analysis, please see [34].
Our work here sketches a method to make use of a more complex structure in goal functions. Of course, we still depend on the right choice of the subgoal predicate g. We suggest that future work should explore methods to generate suitable subgoal predicates g dynamically while we train on the problem domain; thus, we would render the goal function self-adaptive and more strongly connect the results here to the body of work of this thesis.

[9]A tetromino is a shape made up of four squares connected at one of their sides (at least). These shapes are the blocks used to play the game of Tetris.

7. Conclusion

Ἀνθρώποισι πᾶσι μέτεστι γιγνώσκειν ἑαυτοῦς καὶ σωφρονεῖν.
All humans have in common that they learn about themselves and think with reason.

Heraclitus[1]

7.1. Summary

In the course of this thesis we examined the phenomenon of self-adaptive fitness in evolutionary processes. We started from an initial observation that evolutionary processes can actually perform better with respect to their target function t when they use a fitness function $f \neq t$. We examined a major example of how to construct such adjusted fitness functions: diversity. Various diversity measures exist for different purposes but with the same effect to make the fitness of the individual depend on its relation (similarity, e.g.) to other individuals in the population. This makes the fitness function self-adaptive. It may look like this self-adaptiveness makes the optimization process more complex but, following our initial observation, the optimization result may actually improve. We argue that this happens because a self-adaptive fitness can better approximate the ideal trade-off between exploration and exploitation during evolution, which naturally shifts from leaning towards exploration to leaning towards exploitation. We defined the notion of productive fitness to describe the ideal fitness value we would have wanted to assign to an individual, but can only approximate in hindsight—if at all.
We analyze co-evolution as a powerful tool that can generate self-adaptive fitness evaluations. We see that we can even pit various kinds of evolutionary processes against each other and that the joint target function provides a simple interface for their intuitive interaction. A natural arms race then builds suitable curricula for both involved processes. We can generalize these principles to patterns for the design of complex self-adaptive systems and thereby use the phenomenon of self-adaptive fitness not as a side effect but as a design tool that we implement into our processes deliberately.
We also took a look at the other end of the complexity scale from a software engineering perspective: Artificial chemistry systems are built upon very simple rules and may then generate complex behavior on their own. We introduce a new type of artificial chemistry

[1]This quote is found in Stobaeus's *Selections* (3.5.6).

system that uses neural networks that can directly influence each other's weights. We show in accordance with other types of artificial chemistry systems that we can achieve a drive towards a certain robustness that shows in networks that can self-replicate. That drive exists without any externally given fitness function and we can thus consider it an instance of inherent self-adaptive fitness.

Throughout all the research presented here we made a conscious effort to integrate all kinds of phenomena and definitions into a complete and sound formal framework. Where the papers attached naturally show some deviations from a common language, mainly because they were written over the course of a few years, we used this text part of the thesis to unify the vocabulary and build a comprehensive formal framework.

Additionally, we attempted to point out the major contributions throughout this text but needed to leave out many explanations from the papers that can be found in the appendix. We also refer to their respective sections on related work for a broader overview of the literature for all relevant topics. We now return to research questions originally formulated in the introduction of this text (Chapter 1) and do our best to give short and concise answers.

(RQ1) Which phenomena can be described by evolutionary processes? In Chapter 2 we provide a comprehensive formal framework for evolutionary processes in general and evolutionary algorithms specifically. We build a formal framework for artificial chemistry systems as an instance of evolution onto the same foundation in Chapter 3. We augment this framework to encompass reinforcement learning and co-evolutionary processes in Chapter 5. We even show how the initial example of software evolution might fit the given definition of evolutionary processes in a meaningful and useful way (also Chapter 5).

(RQ2) Which phenomena can be described as self-adaptive fitness? We can give a formal definition based on our formal framework (Chapter 2). We explore artificial chemistry systems, which naturally have no predetermined fitness at all but tend to show properties of striving towards a very specific goal, depending on the particles that are currently present in the system. We recognize this as an early, emergent form of self-adaptive fitness. Diversity-aware evolutionary algorithms use a very direct form of self-adaptive fitness as they use a fitness function that directly relies on population-based measurements that naturally change with each generation depending on the results of the algorithm itself (Chpater 4). In Chapter 4 we further see that the freshly introduced productive fitness (or at least its approximation) is self-adaptive by definition. We discover even more powerful means of constructing self-adaptive fitness functions when we analyze co-evolution, where a different evolutionary process is responsible for changes in the self-adaptive joint fitness function (Chapter 5).

(RQ3) Are there goal-independent parts to fitness? In Chapter 3 we see clearly that even systems without a given goal exhibit behavior as if they had an inherent fitness. These tendencies are goal-independent by definition. Diversity-aware evolutionary algorithms in our definition add parts to the fitness function that measure and favor diversity; these parts do not directly depend on the target function and can be formulated in a rather general way that works for many different target functions at least (Chapter 4). The construction of productive fitness (Chapter 4) is also parametric on the target function, but here the respective contributions of the target function and the productive fitness construction are much harder to separate (also Chapter 4).

(RQ4) Is there some inherent goal fitness in evolutionary algorithms as well? We recognize that there are some patterns that most evolutionary algorithms will want to follow, most prominently a suitable balance between exploration and exploitation in the search process. This means that methods like diversity-awareness work for many if not most typical applications of evolutionary algorithms (Chapter 4). However, they are still subject to the No Free Lunch theorem and are thus outperformed in other instances (like target functions that can easily be optimized greedily). In contrast to that, we define productive fitness to be exactly the inherently best choice for a fitness function given a specific target function. However, we gain this purely mathematical advantage by following a purely theoretical construction that is computationally infeasible in practical applications. We can thus say that there indeed is some ideal goal, but there is no direct way to reach it (yet).

(RQ5) How can we discover goal functions for evolutionary processes to follow? In Chapter 4 and the corresponding papers, we discuss many variants of goal function adjustments (such as diversity-awareness) for evolutionary processes. Still, most of these require some kind of assumptions or knowledge on the designer's side, requiring software development on a case-by-case basis. In literature, co-evolution has been discovered as a powerful tool to automatically adjust parts of an evolutionary process as it is evolving. In contrast to many of these methods, such as memetic algorithms, we construct an instance of co-evolutionary processes in Chapter 5 that is based on a very restricted interface: The involved processes only interact via a joint target and fitness function. This allows us to find suitable parameters for fitness functions pretty effectively, as we have shown for our scenario co-evolution approach.

(RQ6) Are there similar phenomena in other kinds of evolutionary processes? Chapter 6 contains a quick overview of research that was inspired by the ongoing search for good self-adaptive fitness functions. We ended up covering further topics of evolutionary computing, quantum computing and planning. Throughout the thesis we make various connections to artificial intelligence, where current methods are based around optimization as well, facing similar problems like the exploration/exploitation dilemma.

7.2. Discussion

Naturally, we point out weaknesses and threats to validity of the studies that form this thesis in their respective papers. Still, we can use this section to discuss the most important shortcoming of the overarching research agenda that we sketched in the text part of this thesis.

The main validation for our formal framework (mostly presented in Chapter 2) is its internal soundness, which is a very formal criterion, and the amount of real-world phenomena we can embed within the framework in a complete and natural manner, which is a very subjective criterion. The text part of this thesis might have made a stronger point here than the individual papers [37, 35] as we subsume a much more diverse set of phenomena under the umbrella of evolutionary processes here, which is why the formal framework is kind of emphasized in the text, compared to the empirical results which we largely delegate to the papers in the appendix. A better indicator for a useful formal framework, however, would be productive power, i.e., if we could produce any theorems from its assumptions. We did apply (parts of) the framework in [33] to sketch a few tools for abstract proofs on evolutionary algorithms, but their potential and validity are not fully explored yet. They just work for the instances where we needed them to. It should also be noted that we sketch a definition of reinforcement learning as an instance of an evolutionary process in Section 5.1 but lose much of the specific features of evolutionary processes when doing so, building but a formal congruence. For a larger approach, the formal framework may be expanded to become more sensitive to the specific difference between evolution and (reinforcement) learning.

In Chapter 3 we also introduce artificial chemistry systems as a specific instance of evolutionary processes, which is arguably a much better fit. We discuss in the text that fitness or fitness-like properties can emerge in artificial chemistry systems and thereby imply that they can form a kind of proto-evolutionary algorithm. The main incentive to make this connection, however, is by analogy to how biological processes (like evolution) emerged from chemical processes (like the primordial soup). It would have been nice to also make that deduction formally and show how an evolutionary algorithm can arise as an abstraction from complex particle interactions. This would have completed the trace of self-adaptive fitness that we follow, but was regarded of little practical concern for the phenomena we wanted to show.

It should be noted that beyond Chapter 3 we never again doubt the validity of the target function. This makes sense as we are concerned about the internal workings of evolutionary algorithms and must assume the target function as given at least at some level. Nonetheless, the field of artificial intelligence shows how much research can go into finding the right way to formulate the programmer's desires and especially for practical optimization algorithms, finding the right target is often surprisingly hard. Of course, there exist methods to break the rule of the one all-knowing target function and even using such a considerably trivial approach as opting for multi-objective evolutionary processes may interact more smoothly with a human-driven design process. For this thesis, we left out all aspects of finding external goals or handling user interaction (even when discussing a recommendation system!), all of which we can only leave to future work.

Our version of co-evolutionary processes (Definition 22, Chapter 5) is rather restricted in that only two processes may be involved with each other and they may only interact via their joint target and/or fitness function. A whole range of more general interaction patterns exist and are discussed in literature. We do not perform any studies on the obvious case of two evolutionary algorithms forming a co-evolutionary process. This is mainly because co-evolution within the realm of evolutionary algorithms has been studied extensively in literature, including very simple forms of co-evolution like island models [80] up to very complex forms like memetic algorithms [61]. Still, our argumentative chain stumbles over the lack of a clean-cut example of only evolutionary algorithms participating in co-evolution. Nonetheless, we would like to argue that the combination of reinforcement learning and evolutionary algorithms is a both a particularly current and a particularly promising field of research.

Finally we feel that possible connections to other fields within natural computing, such as quantum computing (mainly quantum annealing and other optimization algorithms), planning and artificial intelligence in general are somewhat under-utilized in Chapter 6. However, as these topics are not the main focus of this thesis anyway, the holes we leave here may just provide good opportunities for future research.

7.3. Outlook

As we just pointed out in the previous section that the connection of reinforcement learning with the methods of evolutionary algorithms seems rather promising, it is unsurprising that we suggest future work to be done in that direction of research. Most prominently, we see the need to generalize the concept of productive fitness. The research on reward engineering for reinforcement learning shows that finding the right reward function for a given target is just as hard as finding the right fitness function. Formulating the ideal reward function, even if it is infeasible to compute, may aid the search process for a suitable reward.

Even for standard evolutionary algorithms, we see great potential in the automatic construction of fitness functions. We attempted first experiments to automatically adjust the diversity weighting parameter of the fitness function but without finding a stable state of balance. Of course, self-adaptive fitness functions without checks and balances are notorious for finding borderline settings that circumvent solving the computationally hard optimization problem at the heart of the algorithm. Nonetheless, lightweight methods of co-evolution may be able to provide the suitable counterweight.

Artificial chemistry systems also show some properties of self-regulation that might yet be copied in a direct manner to evolutionary algorithms. In most cases, they fall flat due to the abstraction from particle mechanics and usage of population mechanics. Finding

some more connections between these two models may not only benefit evolutionary algorithms but also our neural soup: In preliminary research we could show that the neural networks can be used to perform additional tasks aside from just self-replication. However, we have yet to find a way to utilize a soup of networks to execute a reasonably complex task in a coordinated manner.
Finally, we would like to suggest the hybridization of techniques not only from a software but also from a hardware perspective. Recent research has shown a promising method to intertwine an evolutionary algorithm with quantum annealing [53]. As it is known that evolutionary algorithms benefit from having multiple variants of their operators available [74], mixing in operators that are grounded in entirely different search processes seems like an interesting way to augment evolutionary algorithms and bring other search processes (like quantum annealing) to broad fruition. This way, evolutionary processes provide an open framework for various methods to built upon and can manage the integration of various components via self-adaptation.

Bibliography

[1] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.

[2] Wolfgang Banzhaf and Lidia Yamamoto. *Artificial chemistries.* MIT Press, 2015.

[3] André Baresel, Harmen Sthamer, and Michael Schmidt. Fitness function design to improve evolutionary structural testing. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, pages 1329–1336, 2002.

[4] Christian Bauckhage. Insights into internet memes. In *ICWSM*, pages 42–49, 2011.

[5] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

[6] Susan Blackmore and Susan J Blackmore. *The meme machine*, volume 25. Oxford Paperbacks, 2000.

[7] Tomas Bures, Danny Weyns, Christian Berger, Stefan Biffl, Marian Daun, Thomas Gabor, David Garlan, Ilias Gerostathopoulos, Christine Julien, Filip Krikava, et al. Software engineering for smart cyber-physical systems–towards a research agenda: Report on the first international workshop on software engineering for smart cps. *ACM SIGSOFT Software Engineering Notes*, 2015.

[8] Tomas Bures, Danny Weyns, Bradley Schmer, Eduardo Tovar, Eric Boden, Thomas Gabor, Ilias Gerostathopoulos, Pragya Gupta, Eunsuk Kang, Alessia Knauss, et al. Software engineering for smart cyber-physical systems: challenges and promising solutions. *ACM SIGSOFT Software Engineering Notes*, 2017.

[9] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Software Patterns.* John Wiley & Sons, 1996.

[10] Matej Črepinšek, Shih-Hsi Liu, and Marjan Mernik. Exploration and exploitation in evolutionary algorithms: A survey. *ACM computing surveys (CSUR)*, 45(3):1–33, 2013.

[11] Charles Darwin. *The origin of species.* PF Collier & son New York, 1909.

[12] Richard Dawkins. *The selfish gene.* Oxford university press, 2016.

[13] Richard Dawkins et al. *The blind watchmaker: Why the evidence of evolution reveals a universe without design.* WW Norton & Company, 1996.

[14] Daniel C Dennett. *From bacteria to Bach and back: The evolution of minds.* WW Norton & Company, 2017.

[15] Jason G Digalakis and Konstantinos G Margaritis. On benchmarking functions for genetic algorithms. *International journal of computer mathematics*, 77(4):481–506, 2001.

[16] Peter Dittrich, Jens Ziegler, and Wolfgang Banzhaf. Artificial chemistries—a review. *Artificial life*, 7(3):225–275, 2001.

[17] Agoston E Eiben, Zbigniew Michalewicz, Marc Schoenauer, and James E Smith. Parameter control in evolutionary algorithms. In *Parameter setting in evolutionary algorithms*, pages 19–46. Springer, 2007.

[18] Agoston E Eiben and Cornelis A Schippers. On evolutionary exploration and exploitation. *Fundamenta Informaticae*, 35(1-4):35–50, 1998.

[19] Manfred Eigen and Peter Schuster. The hypercycle. *Naturwissenschaften*, 65(1):7–41, 1978.

[20] Andries P Engelbrecht. *Computational intelligence: an introduction.* John Wiley & Sons, 2007.

[21] Raziyeh Farmani and Jonathan A Wright. Self-adaptive fitness formulation for constrained optimization. *IEEE transactions on evolutionary computation*, 7(5):445–455, 2003.

[22] Walter Fontana and Leo W Buss. What would be conserved if "the tape were played twice"? *Proceedings of the National Academy of Sciences*, 91(2):757–761, 1994.

[23] Walter Fontana and Leo W Buss. "the arrival of the fittest": Toward a theory of biological organization. *Bulletin of Mathematical Biology*, 56(1):1–64, 1994.

[24] Walter Fontana and Leo W Buss. The barrier of objects: from dynamical systems to bounded organizations. 1996.

[25] Thomas Gabor and Philipp Altmann. Benchmarking surrogate-assisted genetic recommender systems. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1568–1575, 2019.

[26] Thomas Gabor and Lenz Belzner. Genealogical distance as a diversity estimate in evolutionary algorithms. In *Measuring and Promoting Diversity in Evolutionary Algorithms (MPDEA@GECCO).* ACM, 2017.

[27] Thomas Gabor, Lenz Belzner, Marie Kiermeier, Michael Till Beck, and Alexander Neitz. A simulation-based architecture for smart cyber-physical systems. In *The International Workshop on Models@run.time for Self-Aware Computing Systems*, 2016.

[28] Thomas Gabor, Lenz Belzner, and Claudia Linnhoff-Popien. Inheritance-based diversity measures for explicit convergence control in evolutionary algorithms. In *The Genetic and Evolutionary Computation Conference (GECCO)*, 2018.

[29] Thomas Gabor, Lenz Belzner, Thomy Phan, and Kyrill Schmid. Preparing for the unexpected: Diversity improves planning resilience in evolutionary algorithms. In *15th IEEE International Conference on Autonomic Computing (ICAC)*, 2018.

[30] Thomas Gabor, Sebastian Feld, Hila Safi, Thomy Phan, and Claudia Linnhoff-Popien. Insights on training neural networks for QUBO tasks. In *First International Workshop on Quantum Software Engineering (Q-SE 2020)*, 2020.

[31] Thomas Gabor, Steffen Illium, Andy Mattausch, Lenz Belzner, and Claudia Linnhoff-Popien. Self-replication in neural networks. In *Artificial Life Conference Proceedings*, pages 424–431. MIT Press, 2019.

[32] Thomas Gabor, Marie Kiermeier, Andreas Sedlmeier, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, and Jan Wieghardt. Adapting quality assurance to adaptive systems: The scenario coevolution paradigm. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2018.

[33] Thomas Gabor and Claudia Linnhoff-Popien. A formal model for reasoning about the ideal fitness in evolutionary processes. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2020.

[34] Thomas Gabor, Jan Peter, Thomy Phan, Christian Meyer, and Claudia Linnhoff-Popien. Subgoal-based temporal abstraction in Monte-Carlo tree search. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 5562–5568. AAAI Press, 2019.

[35] Thomas Gabor, Thomy Phan, and Claudia Linnhoff-Popien. Productive fitness in diversity-aware evolutionary algorithms. *Natural Computing*, pages 1–14, 2021.

[36] Thomas Gabor, Andreas Sedlmeier, Marie Kiermeier, Thomy Phan, Marcel Henrich, Monika Pichlmair, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, and Jan Wieghardt. Scenario co-evolution for reinforcement learning on a

grid world smart factory domain. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 898–906, 2019.

[37] Thomas Gabor, Andreas Sedlmeier, Thomy Phan, Fabian Ritz, Marie Kiermeier, Lenz Belzner, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, Marc Zeller, and Claudia Linnhoff-Popien. The scenario coevolution paradigm: adaptive quality assurance for adaptive systems. *International Journal on Software Tools for Technology Transfer*, pages 1–20, 2020.

[38] Thomas Gabor, Sebastian Zielinski, Sebastian Feld, Christoph Roch, Christian Seidel, Florian Neukart, Isabella Galter, Wolfgang Mauerer, and Claudia Linnhoff-Popien. Assessing solution quality of 3SAT on a quantum annealing platform. In *International Workshop on Quantum Technology and Optimization Problems*. Springer, 2019.

[39] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, and Design Patterns. Elements of reusable object-oriented software. *Reading: Addison-Wesley*, 1995.

[40] John Grefenstette. Genetic algorithms for changing environments. *Parallel Problem Solving from Nature*, 2:137–144, 1992.

[41] Dylan Hadfield-Menell, Smitha Milli, Pieter Abbeel, Stuart J Russell, and Anca Dragan. Inverse reward design. In *Advances in neural information processing systems*, pages 6765–6774, 2017.

[42] Kuk-Hyun Han and Jong-Hwan Kim. Quantum-inspired evolutionary algorithm for a class of combinatorial optimization. *IEEE transactions on evolutionary computation*, 6(6):580–593, 2002.

[43] Chris Hankin. *An introduction to lambda calculi for computer scientists*. King's College, 2004.

[44] Bettina Heim, Troels F Rønnow, Sergei V Isakov, and Matthias Troyer. Quantum versus classical annealing of ising spin glasses. *Science*, 348(6231):215–217, 2015.

[45] Robert Hinterding, Zbigniew Michalewicz, and Agoston E Eiben. Adaptation in evolutionary computation: A survey. In *Proceedings of 1997 Ieee International Conference on Evolutionary Computation (Icec'97)*, pages 65–69. IEEE, 1997.

[46] Matthias Hölzl and Thomas Gabor. Continuous collaboration: A case study on the development of an adaptive cyber-physical system. In *1st International Workshop on Software Engineering for Smart Cyber-Physical Systems (SEsCPS@ICSE)*, 2015.

[47] Matthias Hölzl and Thomas Gabor. Reasoning and learning for awareness and adaptation. In *Software Engineering for Collective Autonomic Systems - The ASCENS Approach*. Springer, 2015.

[48] Matthias Hölzl and Martin Wirsing. Towards a system model for ensembles. In *Formal Modeling: Actors, Open Systems, Biological Systems*, pages 241–261. Springer, 2011.

[49] Thomas Jansen and Ingo Wegener. Evolutionary algorithms—how to cope with plateaus of constant fitness and when to reject strings of the same fitness. *IEEE Transactions on Evolutionary Computation*, 5(6):589–599, 2001.

[50] Mark W Johnson, Mohammad HS Amin, Suzanne Gildert, Trevor Lanting, Firas Hamze, Neil Dickson, Richard Harris, Andrew J Berkley, Jan Johansson, Paul Bunyk, et al. Quantum annealing with manufactured spins. *Nature*, 473(7346):194–198, 2011.

[51] Tadashi Kadowaki and Hidetoshi Nishimori. Quantum annealing in the transverse ising model. *Physical Review E*, 58(5):5355, 1998.

[52] Stuart A Kauffman et al. *The origins of order: Self-organization and selection in evolution.* Oxford University Press, USA, 1993.

[53] James King, Masoud Mohseni, William Bernoudy, Alexandre Fréchette, Hossein Sadeghi, Sergei V Isakov, Hartmut Neven, and Mohammad H Amin. Quantum-assisted genetic algorithm. *arXiv preprint arXiv:1907.00707*, 2019.

[54] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[55] Rudolf Kruse, Christian Borgelt, Frank Klawonn, Christian Moewes, Georg Ruß, Matthias Steinbrecher, and Pascal Held. *Computational intelligence.* Springer, 2011.

[56] Joel Lehman and Kenneth O Stanley. Exploiting open-endedness to solve problems through the search for novelty. In *ALIFE*, pages 329–336, 2008.

[57] Tambet Matiisen, Avital Oliver, Taco Cohen, and John Schulman. Teacher-student curriculum learning. *IEEE transactions on neural networks and learning systems*, 2019.

[58] Catherine C McGeoch. Adiabatic quantum computation and quantum annealing: Theory and practice. *Synthesis Lectures on Quantum Computing*, 5(2):1–93, 2014.

[59] Silja Meyer-Nieberg and Hans-Georg Beyer. Self-adaptation in evolutionary algorithms. In *Parameter setting in evolutionary algorithms*, pages 47–75. Springer, 2007.

[60] Mark Moore and Ajit Narayanan. Quantum-inspired computing. *Dept. Comput. Sci., Univ. Exeter, Exeter, UK*, 1995.

[61] Pablo Moscato, Carlos Cotta, and Alexandre Mendes. Memetic algorithms. In *New optimization techniques in engineering*, pages 53–85. Springer, 2004.

[62] Oscar Nierstrasz, Marcus Denker, Tudor Gîrba, Adrian Kuhn, Adrian Lienhard, and David Roethlisberger. Self-aware, evolving eternal systems. 2008.

[63] Oscar Nierstrasz, Marcus Denker, Tudor Gîrba, Adrian Lienhard, and David Röthlisberger. Change-enabled software systems. In *Software-Intensive Systems and New Computing Paradigms*, pages 64–79. Springer, 2008.

[64] Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa. *DNA computing: new computing paradigms*. Springer, 1998.

[65] Edwin Pednault, John A Gunnels, Giacomo Nannicini, Lior Horesh, and Robert Wisnieff. Leveraging secondary storage to simulate deep 54-qubit sycamore circuits. *arXiv preprint arXiv:1910.09534*, 2019.

[66] Carlo Pinciroli, Michael Bonani, Francesco Mondada, and Marco Dorigo. Adaptation and awareness in robot ensembles: Scenarios and algorithms. In *Software Engineering for Collective Autonomic Systems*, pages 471–494. Springer, 2015.

[67] Mariachiara Puviani, Giacomo Cabri, and Franco Zambonelli. A taxonomy of architectural patterns for self-adaptive systems. In *Proceedings of the International C* Conference on Computer Science and Software Engineering*, pages 77–85, 2013.

[68] André Reichstaller, Thomas Gabor, and Alexander Knapp. Mutation-based test suite evolution for self-organizing systems. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2018.

[69] Irmi Sax, Sebastian Feld, Sebastian Zielinski, Thomas Gabor, Claudia Linnhoff-Popien, and Wolfgang Mauerer. Approximate approximation on a quantum annealer. In *Proceedings of the 17th ACM International Conference on Computing Frontiers*, pages 108–117, 2020.

[70] Samuel S Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. A correspondence between random neural networks and statistical field theory. *arXiv preprint arXiv:1710.06570*, 2017.

[71] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[72] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel,

et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[73] Moshe Sipper, Ryan J Urbanowicz, and Jason H Moore. To know the objective is not (necessarily) to know the objective function, 2018.

[74] William M Spears. Adapting crossover in evolutionary algorithms. In *Evolutionary programming*, pages 367–384, 1995.

[75] William M Spears and Kenneth A Jong. *The role of mutation and recombination in evolutionary algorithms.* George Mason University Fairfax, VA, 1998.

[76] Giovanni Squillero and Alberto Tonda. Divergence of character and premature convergence: A survey of methodologies for promoting diversity in evolutionary optimization. *Information Sciences*, 329:782–799, 2016.

[77] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction.* MIT press, 2018.

[78] Simon Thompson. *Type theory and functional programming.* Addison Wesley, 1991.

[79] Marco Tomassini. Evolutionary algorithms. In *Towards Evolvable Hardware*, pages 19–47. Springer, 1996.

[80] Marco Tomassini. *Spatially structured evolutionary algorithms: Artificial evolution in space and time.* Springer, 2006.

[81] Alan M. Turing. Computing machinery and intelligence. *Mind*, 59(236):433, 1950.

[82] Eiji Uchibe, Masakazu Yanase, and Minoru Asada. Behavior generation for a mobile robot based on the adaptive fitness function. *Robotics and Autonomous Systems*, 40(2-3):69–77, 2002.

[83] Rasmus K Ursem. Diversity-guided evolutionary algorithms. In *International Conference on Parallel Problem Solving from Nature*, pages 462–471. Springer, 2002.

[84] Tom Vodopivec, Spyridon Samothrakis, and Branko Ster. On monte carlo tree search and reinforcement learning. *Journal of Artificial Intelligence Research*, 60:881–936, 2017.

[85] Mark Wineberg and Franz Oppacher. The underlying similarity of diversity measures used in evolutionary computation. In *Genetic and Evolutionary Computation Conference*, pages 1493–1504. Springer, 2003.

[86] Christian Wirth, Riad Akrour, Gerhard Neumann, and Johannes Fürnkranz. A survey of preference-based reinforcement learning methods. *The Journal of Machine Learning Research*, 18(1):4945–4990, 2017.

[87] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

[88] Ivan Zelinka, Roman Senkerik, and Michal Pluhacek. Do evolutionary algorithms indeed require randomness? In *2013 IEEE Congress on Evolutionary Computation*, pages 2283–2289. IEEE, 2013.

A. Papers

In the appendix we attach all 15 papers that contain the core findings of this thesis. In Section A.1 on the following double page 62–63 we provide a concise overview of the used papers in the order they are originally mentioned in Section 1.3. We also reprint the Table 1.1 below for easier reference. We go into more detail regarding the core contributions of these papers and the credit for them in Section A.2 as it is required by the regulations of faculty 16 at LMU Munich. Finally, we append all papers in full text (Section A.3, pages 71–239).

thesis chapter	research questions	main papers
2. Foundations of Evolutionary Processes	RQ1, RQ2	[33]
3. Emergence of Fitness	RQ3	[31]
4. The Ideal Fitness	RQ2, RQ4	[26, 28, 29, 35]
5. Co-Evolutionary Adaptation of Fitness	RQ5	[27, 32, 36, 37]
6. Applications in Natural Computing	RQ6	[25, 68, 38, 30, 34]

A.1. Overview

Attached papers with their number within the fifteen papers, their reference number in the bibliography, and their main chapter. Full texts can be found at the given page.

No.	Ref.	Ch.	Title	Page
1	[33]	2	Thomas Gabor and Claudia Linnhoff-Popien. A formal model for reasoning about the ideal fitness in evolutionary processes. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2020	71
2	[31]	3	Thomas Gabor, Steffen Illium, Andy Mattausch, Lenz Belzner, and Claudia Linnhoff-Popien. Self-replication in neural networks. In *Artificial Life Conference Proceedings*, pages 424–431. MIT Press, 2019	90
3	[26]	4	Thomas Gabor and Lenz Belzner. Genealogical distance as a diversity estimate in evolutionary algorithms. In *Measuring and Promoting Diversity in Evolutionary Algorithms (MPDEA@GECCO).* ACM, 2017	98
4	[28]	4	Thomas Gabor, Lenz Belzner, and Claudia Linnhoff-Popien. Inheritance-based diversity measures for explicit convergence control in evolutionary algorithms. In *The Genetic and Evolutionary Computation Conference (GECCO)*, 2018	104
5	[29]	4	Thomas Gabor, Lenz Belzner, Thomy Phan, and Kyrill Schmid. Preparing for the unexpected: Diversity improves planning resilience in evolutionary algorithms. In *15th IEEE International Conference on Autonomic Computing (ICAC)*, 2018	112
6	[35]	4	Thomas Gabor, Thomy Phan, and Claudia Linnhoff-Popien. Productive fitness in diversity-aware evolutionary algorithms. *Natural Computing*, pages 1–14, 2021	122
7	[27]	5	Thomas Gabor, Lenz Belzner, Marie Kiermeier, Michael Till Beck, and Alexander Neitz. A simulation-based architecture for smart cyber-physical systems. In *The International Workshop on Models@run.time for Self-Aware Computing Systems*, 2016	136
8	[32]	5	Thomas Gabor, Marie Kiermeier, Andreas Sedlmeier, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, and Jan Wieghardt. Adapting quality assurance to adaptive systems: The scenario coevolution paradigm. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2018	142

No.	Ref.	Ch.	Title	Page
9	[36]	5	Thomas Gabor, Andreas Sedlmeier, Marie Kiermeier, Thomy Phan, Marcel Henrich, Monika Pichlmair, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, and Jan Wieghardt. Scenario co-evolution for reinforcement learning on a grid world smart factory domain. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 898–906, 2019	159
10	[37]	5	Thomas Gabor, Andreas Sedlmeier, Thomy Phan, Fabian Ritz, Marie Kiermeier, Lenz Belzner, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, Marc Zeller, and Claudia Linnhoff-Popien. The scenario coevolution paradigm: adaptive quality assurance for adaptive systems. *International Journal on Software Tools for Technology Transfer*, pages 1–20, 2020	168
11	[25]	6	Thomas Gabor and Philipp Altmann. Benchmarking surrogate-assisted genetic recommender systems. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1568–1575, 2019	188
12	[68]	6	André Reichstaller, Thomas Gabor, and Alexander Knapp. Mutation-based test suite evolution for self-organizing systems. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2018	197
13	[38]	6	Thomas Gabor, Sebastian Zielinski, Sebastian Feld, Christoph Roch, Christian Seidel, Florian Neukart, Isabella Galter, Wolfgang Mauerer, and Claudia Linnhoff-Popien. Assessing solution quality of 3SAT on a quantum annealing platform. In *International Workshop on Quantum Technology and Optimization Problems*. Springer, 2019	214
14	[30]	6	Thomas Gabor, Sebastian Feld, Hila Safi, Thomy Phan, and Claudia Linnhoff-Popien. Insights on training neural networks for QUBO tasks. In *First International Workshop on Quantum Software Engineering (Q-SE 2020)*, 2020	227
15	[34]	6	Thomas Gabor, Jan Peter, Thomy Phan, Christian Meyer, and Claudia Linnhoff-Popien. Subgoal-based temporal abstraction in Monte-Carlo tree search. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 5562–5568. AAAI Press, 2019	233

A.2. Credit

Numbers according to the overview in Section A.1. We give a short summary of the contributions of the paper and by the authors, respectively, in accordance with LMU Munich regulations. We also mention where in the text the paper is mainly treated.

No.	1
Paper	[33] Thomas Gabor and Claudia Linnhoff-Popien. A formal model for reasoning about the ideal fitness in evolutionary processes. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2020
Main Contributions	(i) formal framework for evolutionary algorithms; (ii) preliminary landscape analysis of productive fitness; (iii) preliminary tools for proofs about fitness
Credit	Gabor conceived the main contributions, performed the experiments and produced the formal arguments. Linnhoff-Popien consulted the process and reviewed the results.
Treatment	main focus of Chapter 2 (RQ1, RQ2)

No.	2
Paper	[31] Thomas Gabor, Steffen Illium, Andy Mattausch, Lenz Belzner, and Claudia Linnhoff-Popien. Self-replication in neural networks. In *Artificial Life Conference Proceedings*, pages 424–431. MIT Press, 2019
Main Contributions	(i) methods for self-applicable neural networks; (ii) self-stabilizing neural network particle soup (concept and empirical analysis); (iii) non-trivial fixpoints wrt. self-application for neural networks
Credit	Gabor conceived the original concept, formal notation, and the methods of analysis; he also conducted the core experiments. Illium co-conceived the advanced concepts, supported practical experiments and performed further analysis on the results. Mattausch performed preliminary experiments (including conceiving the means to implement them) to this paper as a practical course. Mattausch and Belzner provided discussion and ideas for the original concept. Linnhoff-Popien consulted the process and reviewed the results.
Treatment	main focus of Chapter 3 (RQ3)

No.	3
Paper	[26] Thomas Gabor and Lenz Belzner. Genealogical distance as a diversity estimate in evolutionary algorithms. In *Measuring and Promoting Diversity in Evolutionary Algorithms (MPDEA@GECCO)*. ACM, 2017
Main Contributions	(i) novel domain-independent measurement method for diversity in evolutionary algorithms (concept and empirical analysis); (ii) analysis of random approximation for all-pairs diversity measurements
Credit	Gabor conceived the original concepts and conducted the empirical analysis. Belzner discussed and reviewed the process and the results.
Treatment	main focus of Chapter 4 (RQ2)

No.	4
Paper	[28] Thomas Gabor, Lenz Belzner, and Claudia Linnhoff-Popien. Inheritance-based diversity measures for explicit convergence control in evolutionary algorithms. In *The Genetic and Evolutionary Computation Conference (GECCO)*, 2018
Main Contributions	(i) concise formal framework for inheritance-based diversity; (ii) extended analysis of various methods of diversity (incl. genealogical)
Credit	Gabor conceived the original concepts and conducted the empirical analysis. Belzner discussed and reviewed the process and the results. Linnhoff-Popien reviewed the process and the results.
Treatment	main focus of Chapter 4 (RQ2)

No.	5
Paper	[29] Thomas Gabor, Lenz Belzner, Thomy Phan, and Kyrill Schmid. Preparing for the unexpected: Diversity improves planning resilience in evolutionary algorithms. In *15th IEEE International Conference on Autonomic Computing (ICAC)*, 2018
Main Contributions	connection between dynamic optimization and diversity (advanced concept, practical application, and empirical analysis)
Credit	Gabor conceived the original concepts and conducted the empirical analysis. Belzner, Phan, and Schmid discussed and reviewed the process and the results.
Treatment	main focus of Chapter 4 (RQ2)

No.	6
Paper	[35] Thomas Gabor, Thomy Phan, and Claudia Linnhoff-Popien. Productive fitness in diversity-aware evolutionary algorithms. *Natural Computing*, pages 1–14, 2021
Main Contributions	(i) novel concept of productive fitness (incl. formalization and empirical analysis); (ii) connection between adjusted fitness functions (focus on diversity) and productive fitness
Credit	Gabor conceived the original concepts and conducted the empirical analysis. Phan supported the development of the concepts, discussed the novel notions and reviewed the process. Linnhoff-Popien reviewed the process and the results.
Treatment	main focus of Chapter 4 (RQ4)

No.	7
Paper	[27] Thomas Gabor, Lenz Belzner, Marie Kiermeier, Michael Till Beck, and Alexander Neitz. A simulation-based architecture for smart cyber-physical systems. In *The International Workshop on Models@run.time for Self-Aware Computing Systems*, 2016
Main Contributions	(i) formalization of communication patterns and architectures for the treatment of simulation in adaptive systems (with focus on cyber-physical systems); (ii) concept for the categorization of control systems within an adaptive system
Credit	Gabor conceived the concepts and produced their final form. Belzner co-developed the concepts from their initial version. Kiermeier, Beck, and Neitz discussed and reviewed the results as well as helped with the presentation in the paper.
Treatment	main focus of Chapter 5 (RQ5)

No.	8
Paper	[32] Thomas Gabor, Marie Kiermeier, Andreas Sedlmeier, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, and Jan Wieghardt. Adapting quality assurance to adaptive systems: The scenario coevolution paradigm. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2018
Main Contributions	*These contributions are repeated in their enirety in the substantially extended version No. 10* [37]. (i) extension of the GEM framework [48] for optimization; (ii) formalization of the software development process as co-evolution; (iii) discovery of patterns in the design of said co-evolution
Credit	Gabor conceived the concepts and produced their final form. Kiermeier, Sedlmeier, Kempter, Klein, Sauer, Schmid, and Wieghardt reviewed and discussed the concepts intensively and provided various ideas.
Treatment	main focus of Chapter 5 (RQ5)

No.	9
Paper	[36] Thomas Gabor, Andreas Sedlmeier, Marie Kiermeier, Thomy Phan, Marcel Henrich, Monika Pichlmair, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, and Jan Wieghardt. Scenario co-evolution for reinforcement learning on a grid world smart factory domain. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 898–906, 2019
Main Contributions	(i) co-evolutionary genetic algorithm for auto-curriculum generation in reinforcement learning (concept and empirical analysis); (ii) generation of objectively hard test cases as a by-product (concept and empirical analysis)
Credit	Gabor conceived the concepts, defined the algorithm, the goals, and the methods of analysis, co-supervised the development of the implementation and co-performed the analysis if the data. Sedlmeier, Kiermeier, and Phan co-supervised the implementation and provided various bits of input and discussion. Sedlmeier also co-performed the analysis. Henrich and Pichlmair implemented the algorithm and provided the setup for experiments as a practical course. Kempter, Klein, Sauer, Schmid, and Wieghardt reviewed and discussed the concepts intensively and provided various ideas.
Treatment	main focus of Chapter 5 (RQ5)

No.	10
Paper	[37] Thomas Gabor, Andreas Sedlmeier, Thomy Phan, Fabian Ritz, Marie Kiermeier, Lenz Belzner, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, Marc Zeller, and Claudia Linnhoff-Popien. The scenario coevolution paradigm: adaptive quality assurance for adaptive systems. *International Journal on Software Tools for Technology Transfer*, pages 1–20, 2020
Main Contributions	*This is a substantially extended version of No. 8* [32]. (iv) conceptualization and software-engineering treatment of a machine learning pipeline; (v) formalization and contextualization of the practical experiments of [36]
Credit	For the additional contributions from the extension: Gabor applied and integrated the machine learning pipeline with the formal framework; he conceived, formalized, contextualized, and executed the practical experiments. Belzner conceived the machine learning pipeline. Sedlmeier, Phan, Ritz, Kiermeier, Belzner, Kempter, Klein, Sauer, Schmid, Wieghardt, Zeller, and Linnhoff-Popien reviewed and discussed the concepts intensively and provided various ideas.
Treatment	main focus of Chapter 5 (RQ5)

No.	11
Paper	[25] Thomas Gabor and Philipp Altmann. Benchmarking surrogate-assisted genetic recommender systems. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1568–1575, 2019
Main Contributions	(i) surrogate-based optimization "without replacement" (concept and empirical analysis); (ii) preliminary proof-of-concept for the application of said method for recommendation systems
Credit	Gabor co-conceived the concepts; he steered the research process, supervised the work, and reviewed the results. Altmann co-conceived the concepts; he developed them to implementation and performed the experiments. This paper resulted from Altmann's thesis to achieve the degree Bachelor of Science.
Treatment	main focus of Chapter 6 (RQ6)

No.	12
Paper	[68] André Reichstaller, Thomas Gabor, and Alexander Knapp. Mutation-based test suite evolution for self-organizing systems. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2018
Main Contributions	(i) formalization and test design for testing self-organizing systems with a mutation-based test goal (concept and empirical analysis); (ii) novel extensions to evolutionary algorithms for graph-based and set-based search spaces (concepts and empirical analysis)
Credit	Reichstaller and Knapp conceived the overall study, the concepts and formalization for the mutation-based testing of the considered instance of a self-organizing system-under-test. Reichstaller implemented the system-under-test and the generation of test mutants and provided empirical data. Gabor conceived and implemented the evolutionary algorithm and its extensions and performed the respective experiments. Knapp also reviewed and discussed the results.
Treatment	main focus of Chapter 6 (RQ6)

No.	13
Paper	[38] Thomas Gabor, Sebastian Zielinski, Sebastian Feld, Christoph Roch, Christian Seidel, Florian Neukart, Isabella Galter, Wolfgang Mauerer, and Claudia Linnhoff-Popien. Assessing solution quality of 3SAT on a quantum annealing platform. In *International Workshop on Quantum Technology and Optimization Problems*. Springer, 2019
Main Contributions	empirical analysis of quantum annealing behavior for 3SAT (incl. initial connection between problem difficulty and solution result for quantum annealing for 3SAT problems)
Credit	Gabor co-conceived the concepts; he steered the research process, supervised the work, and reviewed the results. Zielinski co-conceived the concepts, implemented the software and performed the experiments. Feld, Seidel and Mauerer provided ideas for analysis. Feld, Roch, Seidel, Neukart, Galter, Mauerer, and Linnhoff-Popien reviewed and discussed the concepts intensively and provided various ideas as well as practical support.
Treatment	main focus of Chapter 6 (RQ6)

No.	14
Paper	[30] Thomas Gabor, Sebastian Feld, Hila Safi, Thomy Phan, and Claudia Linnhoff-Popien. Insights on training neural networks for QUBO tasks. In *First International Workshop on Quantum Software Engineering (Q-SE 2020)*, 2020
Main Contributions	neural-network-based approximation of optimization, especially quantum annealing (concept of implementation and preliminary empirical results)
Credit	Gabor conceived the original concept and co-constructed the experiments. Gabor and Feld steered the research process, supervised the work, and reviewed the results. Safi co-constructed the experiments, implemented the concepts in software, performed the experiments and provided the empirical data for analysis. Linnhoff-Popien reviewed and discussed the process and the results. This paper resulted from Safi's thesis to achieve the degree Master of Science.
Treatment	main focus of Chapter 6 (RQ6)

No.	15
Paper	[34] Thomas Gabor, Jan Peter, Thomy Phan, Christian Meyer, and Claudia Linnhoff-Popien. Subgoal-based temporal abstraction in Monte-Carlo tree search. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 5562–5568. AAAI Press, 2019
Main Contributions	subgoal-based extension to MCTS (concept and empirical analysis)
Credit	Gabor discussed the initial concept and steered its development; he supervised the work, reviewed the results and discussed their analsysis. Peter provided the initial concept, developed it to implementation, performed the experiments, and provided the initial analysis. Phan discussed and reviewed the results and conceived and executed the analysis and presentation in the paper. Meyer contributed to the initial concept and discussed the approach. Linnhoff-Popien reviewed and discussed the process and the results. This paper resulted from Peter's thesis to achieve the degree Master of Science.
Treatment	main focus of Chapter 6 (RQ6)

A.3. Full Texts

We now append the full texts of the discussed papers as they appear in their publication or on arXiv. We reprint them to adhere to LMU Munich requirements. Please cite them in their original publication format. To fit the layout of this thesis, we slightly scale down the original pages, which allows us to show page numbers for easy reference. Beyond the attached papers, this thesis features no further content.

Taken from original publication: Thomas Gabor and Claudia Linnhoff-Popien. A formal model for reasoning about the ideal fitness in evolutionary processes. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2020

A Formal Model for Reasoning About the Ideal Fitness in Evolutionary Processes

Thomas Gabor(✉) and Claudia Linnhoff-Popien

LMU Munich, Munich, Germany
thomas.gabor@ifi.lmu.de

Abstract. We introduce and discuss a formal model of evolutionary processes that summarizes various kinds of evolutionary algorithms and other optimization techniques. Based on that framework, we present assumptions called "random agnosticism" and "based optimism" that allow for new kinds of proofs about evolution. We apply them by providing all a proof design that the recently introduced notion of final productive fitness is the ideal target fitness function for any evolutionary process, opening up a new perspective on the fitness in evolution.

Keywords: Evolution · Evolutionary algorithms · Fitness

1 Introduction

Evolution in its broadest sense describes a process that finds solutions to complex problems via the application of comparatively simple local operators. Mostly, this process can be described as a search that starts quite uninformed and uses the knowledge gained through trial and error to guide the further search process. Note that usually this happens without central control and mostly without even any central viewpoint that would allow to overlook all parts of the evolution. However, evolution is often implemented deliberately (using evolutionary algorithms in software, e.g.) in order to search or optimize for a specific result according to an externally given target.

While this target is often provided directly to the evolutionary process so that intermediate results may be evaluated, many studies empirically show better results when using slightly different goal than going directly for the external target metric. Our recent study [8] has brought up empirical evidence that one such "indirect" metric (called *final productive fitness*) might be theoretically optimal (even when or perhaps because it is extremely costly to compute). However, little formal framework exists to reason about evolutionary processes (specifically goals in evolutionary processes) at such a broad level in order to formally prove a claim of optimality.

The aim of this paper is to show what kind of formal framework *would be sufficient* to produce a formal proof of final productive's fitness optimality. To this end, we first introduce two bold but crucial assumptions hat allow us to strip

© Springer Nature Switzerland AG 2020
T. Margaria and B. Steffen (Eds.): ISoLA 2020, LNCS 12477, pp. 473–490, 2020.
https://doi.org/10.1007/978-3-030-61470-6_28

Taken from original publication: Thomas Gabor and Claudia Linnhoff-Popien. A formal model for reasoning about the ideal fitness in evolutionary processes. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2020

away much of the complexity of reasoning about evolution. Then we construct the desired proof from them to show how they work. We hope that the novel tools (i.e., mainly Assumptions 1 and 2) designed here can be used for other high-level arguments about evolution.

All necessary definitions involving evolutionary processes are given in a consistent way in Sect. 2. Section 3 then discusses the issue of the ideal fitness function and introduces the tools to reason about it. We give a short glance at related work in Sect. 4 and conclude with Sect. 5.

2 Definitions

We follow the formal framework sketched in [8] to a vast extent but substantially expand it in generality. We provide an example in Sect. 2.3.

2.1 Evolutionary Processes

For all definitions, we aim to give them in such a basic form that they can span over various disciplines, from biology to formal methods. We use $\mathfrak{P}$ to denote the power set.

Definition 1 (Evolution)**.** *Let $\mathcal{X}$ be an arbitrary set called* search space*. Let $g \in \mathbb{N}$ be called the* generation count*. Let $X_i \subseteq \mathcal{X}$ for any $i \in \mathbb{N}, 0 \leq i \leq g$ be a subset of $\mathcal{X}$ called* population*. Let $E : \mathfrak{P}(\mathcal{X}) \rightarrow \mathfrak{P}(\mathfrak{P}(\mathcal{X}))$ be a function called* evolutionary function.

A tuple $(\langle X_i \rangle_{0 \leq i \leq g}, E)$ is called an evolution *over $\mathcal{X}$ iff $X_i \in E(X_{i-1})$ for all $i \in \mathbb{N}, 1 \leq i \leq g$.*

Any element of the search space $x \in \mathcal{X}$ is called solution candidate (or sometimes just solution for short). Members of a given population $x \in X$ are obviously always solution candidates, but are often also called individuals. Every i within the generation count $1 \leq i \leq g$ is called a generation number with X_i being the respective generation's population. If no confusion is possible, both i and X_i will also be called a generation. X_0 is called the initial population.

Note that an evolution can be generated given a configuration consisting of a search space $\mathcal{X}$, an initial population X_0 and an evolution function E. However, many possible evolutions can follow from the same configuration. Often, the initial population X_0 is not given as a set but instead generated (semi-)randomly. We write that as an initialization function $I : \mathfrak{R} \rightarrow \mathfrak{P}(\mathcal{X})$ where $\mathfrak{R}$ stands for random inputs.[1] Notation-wise, we omit random inputs and write $X_0 \sim I()$ (or simply $X_0 = I()$ if no confusion is possible) for the initial population generated by such a function.

Definition 2 (Target)**.** *Let $\mathcal{X}$ be a search space. A function $t : \mathcal{X} \rightarrow [0; 1]$ that assigns all elements in the search space a scalar value is called a* target function.

[1] In computers, these are often provided by a seed value and a hash function.

Taken from original publication: Thomas Gabor and Claudia Linnhoff-Popien. A formal model for reasoning about the ideal fitness in evolutionary processes. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2020

A target function assigns a value to each point in the search space, i.e., to any given solution candidate.[2] We assume that target values are bounded, so w.l.o.g. we can assume the target value space to be restricted to $[0;1]$ in Definition 2. Again this can be generalized but is rarely useful in praxis. Also note that target functions themselves are unconstrained: They can always be applied to the whole search space. Hard constraints must be implemented by altering the search space or "softening" them by representing them with different target values.

Furthermore, w.l.o.g. we assign every goal function a minimization semantic: For two solution candidates $x_1, x_2 \in \mathcal{X}$ we say that x_1 fulfills a goal t better iff $t(x_1) < t(x_2)$. Any solution candidate $x \in \mathcal{X}$ so that $t(x) \leq t(x') \quad \forall x' \in \mathcal{X}$ is called a global optimum. An algorithm searching for increasingly better solutions candidates is called an optimization algorithm. A configuration, a target function and an evolution form an evolutionary process:

Definition 3 (Evolutionary Process)**.** *Let $\mathcal{X}$ be a search space. Let $E : \mathfrak{P}(\mathcal{X}) \to \mathfrak{P}(\mathfrak{P}(\mathcal{X}))$ be an evolutionary function. Let $t : \mathcal{X} \to [0;1]$ be a target function. Let X_i be a population for any $i \in \mathbb{N}, 0 \leq i \leq g$.*

A tuple $\mathcal{E} = (\mathcal{X}, E, t, \langle X_i \rangle_{i \leq g})$ is an evolutionary process *iff $(\langle X_i \rangle_{i \leq g}, E)$ is an evolution.*

Effectively, an evolutionary process consists of a history of past populations (X_i) and the means to generate new population (E). We often implement the evolutionary function by giving an *evolutionary step function* $e : \mathfrak{P}(\mathcal{X}) \times \mathfrak{R} \to \mathfrak{P}(\mathcal{X})$ and write $X_{i+1} \sim e(X_i)$ (or simply $X_{i+1} = e(X_i)$ if no confusion is possible) for any population X_{i+1} that evolved from X_i by applying the evolutionary step function alongside with some (omitted) random input.

An evolutionary process also carries a target function t. An evolutionary process $\mathcal{E}$ is *optimizing* iff $\min_{x \in X_0} t(x) \geq \min_{x' \in X_g} t(x')$. For many mechanisms in stochastic search as well as for more natural phenomena like biological evolution or human software development processes, optimization is a rather strong property. However, if we have sufficient space within a population and access to the target function, we can turn all evolutionary processes into optimizing ones by just saving the currently best individual alongside the evolution, i.e., ensuring that $\arg\min_{x \in X_i} t(x) \in X_{i+1}$.

Definition 4 (Elitism)**.** *An evolutionary process $\mathcal{E} = (\mathcal{X}, E, t, \langle X_i \rangle_{i \leq g})$ is called* elitist *iff for all $i \in \mathbb{N}, 1 \leq i \leq g$, it holds that $\min_{x \in X_{i-1}} t(x) \geq \min_{x' \in X_i} t(x')$.*

All elitist processes are optimizing. If not noted differently, we from now on assume every evolutionary process to be elitist by default.

[2] Note that by giving a function only parametrized on the individual itself, we assume that the target function is static. Dynamic optimization is an entire field of research that we heavily use in this paper. However, we leave dynamic target functions in our formalism to future work.

2.2 Evolutionary Algorithms

An evolutionary algorithm is special case of evolutionary process that uses an evolutionary function made up of a number of standard components called *evolutionary operators*. We now introduce standard definitions for these components that most instances of evolutionary algorithms can be mapped to. However, the field of evolutionary algorithms is vast and there are variants that alter many smaller details of how they work. It is interesting to note how robust the general concept of evolution is to such variations.

Nearly all evolutionary algorithms that use set-based populations introduce a fixed population size $n \in \mathbb{N}$ for all generations. This allows to keep memory resources easily manageable as the overall memory consumption will not increase over time. We also use this opportunity to introduce the concept of fitness functions. Note that $\mathfrak{E}$ is the space of all evolutionary processes.

Definition 5 (Fitness)**.** *Let $\mathcal{X}$ be a search space. A function $f : \mathcal{X} \times \mathfrak{E} \times \mathfrak{R} \to [0;1]$ is called a* fitness function. *This function takes an individual, its evolutionary process up until now, and random input and returns a scalar value.*

The fitness function can be regarded as generalization of the concept of a target function (cf. Definition 2). It represents the goal definition that the evolutionary process can call upon and actively follows, which may or may not coincide with the target function. In addition to the solution candidate itself, it is able to process additional information about the context. Various approaches may allow nearly arbitrary information here. For a rather general approach, we just pass on a snapshot of the evolutionary process that generated the individual until now. This includes:

- The current population that the evaluated individual is a part of allows to define the fitness of an individual relative to its peers.
- The history of all populations until now allows to observe relative changes over time as well as trace the ancestry of individuals throughout evolution.
- The number of the current generation allows the fitness function to change over time and implement, e.g., a cool-down schedule.

Note that the random input that is also passed along allows fitness functions to also vary fitness values stochastically. However, most fitness functions will not make use of all this information. In these cases we allow to trim down the fitness function's signature and simply write $f(x)$ for an individual $x \in \mathcal{X}$ if all other parameters are ignored.

In many practical instances, developers will choose the target function as a fitness function, i.e., $f(x) = t(x)$ for all $x \in \mathcal{X}$, and for most target functions, evolution will end up achieving passable target values this way. It is the main point of this paper, however, to prove that the optimal choice in general is a different function derived from the target function.

Alongside the fitness function f an evolutionary algorithm also uses various selection functions. In general, a selection function returns a subset of the population for a specific purpose.

Taken from original publication: Thomas Gabor and Claudia Linnhoff-Popien. A formal model for reasoning about the ideal fitness in evolutionary processes. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2020

Definition 6 (Selection)**.** *A function* $s : \mathfrak{P}(\mathcal{X}) \times \mathfrak{E} \times \mathfrak{R} \to \mathfrak{P}(\mathcal{X})$ *is called a* selection function *iff* $s(X, \mathcal{E}, r) \subseteq X$ *for all* $X, \mathcal{E}, r$. *This function takes a population, its evolutionary process, and random input and returns a subset of the given population.*

Again note that we allow for a multitude of information that will rarely be used directly in any selection function and that will be omitted if not necessary. Most importantly, however, any selection function is able to call any fitness function since all its inputs can be provided.

As seemingly limitless variations of selection functions exist we use this opportunity to provide a few examples and at the same time define all families of selection functions that we use for the remainder of this paper. (Note that the current population X is always provided with an evolutionary process $\mathcal{E}$.)

Random Selection. This function $\varrho^m(X, \mathcal{E}, r) = \{x \sim X\} \cup \varrho^{m-1}(X, \mathcal{E}, r)$ selects m individuals of the population at random. Note that $x \sim X$ is one element $x \in X$ sampled uniformly at random. We define $\varrho^0(X, \mathcal{E}, r) = \emptyset$.

Cutoff Selection. This function $\sigma^m(X, \mathcal{E}, r) = \{\arg\min_{x \in X} f(x, \mathcal{E}, r)\} \cup \sigma^{m-1}(X, \mathcal{E}, r)$ selects the m best individuals according to the fitness function f. We define $\sigma^0(X, \mathcal{E}, r) = \emptyset$.

We can now move on to define the evolution function E. For all variants of evolutionary algorithms there exist certain building blocks, called *evolutionary operators*, that most evolutionary functions have in common. They take as arguments some individuals and return some (possibly new) individuals. During the execution of an evolutionary operator its input individuals are referred to as *parents* and its output individuals are referred to as *children*.

Mutation. This operator $mut : \mathcal{X} \times \mathfrak{R} \to \mathcal{X}$ generates a randomly slightly altered individual from a parent.

Recombination. This operator $rec : \mathcal{X} \times \mathcal{X} \times \mathfrak{R} \to \mathcal{X}$ takes two individuals to combine them into a new individual.

Migration. This operator $mig : \mathfrak{R} \to \mathcal{X}$ generates a random new individual.

Again, countless variants and implementations exist, most importantly among them there is non-random mutation and recombination with various amounts of parents and children. For brevity, we omit everything we do not use in this paper's study. Please note that all of these operators return entirely new individuals and leave their parents unchanged. In practical applications, it is equally common to apply (some of) these operators *in-place*, which means that the generated children replace their parents immediately. We, however, opt to just add the children to the population (and possibly eliminate the parents later) so that parents and their children can exist side by side within the same generation. Our main aim in doing this is that it makes elitism much easier to achieve.

As these operators work on single individuals, we define a shortcut to apply them to sets of individuals:

Taken from original publication: Thomas Gabor and Claudia Linnhoff-Popien. A formal model for reasoning about the ideal fitness in evolutionary processes. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2020

Definition 7 (Application of Operators)**.** *Let $X \subseteq \mathcal{X}$ be a set of individuals in search space $\mathcal{X}$. Let s be a selection function. We write $X \downharpoonleft_{mut} s = \{mut(x) \mid x \in s(X)\}$ and $X \downharpoonleft_{rec} s = \{rec(x_1, x_2) \mid x_1 \in s(X), x_2 \sim X\}$ for the sets of children when applying the respective operators. For consistency, we also write $X \downharpoonleft_{mig} s = \{mig() \mid x \in s(X)\}$ to create $|s(X)|$ many new random individuals, even though their values do not depend on the individuals in X.*

We are now ready to define a scheme for the evolution function E in evolutionary algorithms. We do so by providing an evolutionary step function e as discussed above with parameters $A_1, A_2, A_3 \in \mathbb{N}$:

$$e(X) = \sigma^{|X|}(X \cup (X \downharpoonleft_{rec} \sigma^{A_1}) \cup (X \downharpoonleft_{mut} \varrho^{A_2}) \cup (X \downharpoonleft_{mig} \varrho^{A_3})) \qquad (1)$$

Note again that in this evolutionary step we place all generated children alongside their parents into one population and then cutoff-select the best from this population.[3] As it is common, we use random selection to select mutation parents. The selection function for the recombination parents is also called *parent selection.* We use cutoff selection on one parent with a randomly selected partner here. This gives some selective pressure (i.e., better individuals have a better chance of becoming recombination parents) without overcentralizing too much. Although many approaches to parent selection exist, we choose this one as it is both effective in practical implementations and mathematically very clean to define. The final selection function that is called upon the combined population of potential parents and new children is called *survivor selection.* We simply use cutoff selection here for ease of reasoning. Many evolutionary algorithms use more advanced survivor selection functions like roulette wheel selection where better individuals merely have a higher chance of being picked. We choose a hard cutoff for this kind of selection, mainly because it is simpler to define and understand, and its transparent to elitism. Since the cutoff point varies with the population's fitness structure that is subjected to random effects, the practical difference between both approaches for our examples is negligible. Note that we can emulate a lot of different selection schemes by choosing an appropriate fitness function: As the fitness function can vary at random, we can for example make the cutoff more fuzzy by simply adding noise to each fitness evaluation instead of changing the selection function. Also note that adding all the children non-destructively and using cutoff-selection makes the algorithm elitist if $f = t$.

We parametrize the evolutionary step function with the amount of recombination children A_1, amount of mutation children A_2 and amount of migration children A_3. These are also often given as rates relative to the population size.

Definition 8 (Evolutionary Algorithm)**.** *An evolutionary algorithm is an evolutionary process $\mathbb{E} = (\mathcal{X}, E, t, \langle X_i \rangle_{i \leq g})$ where the evolutionary function is given via an evolutionary step function of the form described in Eq. 1, where a fitness function f is used for all selection functions and evolutionary operators and the target function t is only accessible insofar it is part of f.*

[3] In the field of evolutionary computing, this is called a $\mu + \lambda$ selection scheme.

Taken from original publication: Thomas Gabor and Claudia Linnhoff-Popien. A formal model for reasoning about the ideal fitness in evolutionary processes. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2020

Note that for the ease of use in later notation, we will often denote two evolutionary processes that differ solely in their fitness function (ϕ vs. ψ, e.g.) by denoting that fitness function as a subscript ($\mathcal{E}_\phi$ vs. $\mathcal{E}_\psi$). Independently of that we denote the best individual of the final generation of $\mathcal{E}_\phi$ according to some fitness or target function ψ with

$$|\mathcal{E}_\phi|_\psi = \arg\min_{x \in X_g} \psi(x) \tag{2}$$

and the best of all generations with

$$||\mathcal{E}_\phi||_\psi = \arg\min_{\substack{x \in X_i \\ i \in \mathbb{N} \\ 0 \leq i \leq g}} \psi(x). \tag{3}$$

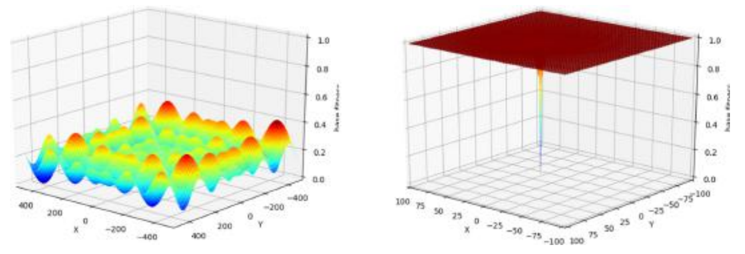
It is clear that if $\mathcal{E}_\phi$ is elitist with respect to ψ, then $|\mathcal{E}_\phi|_\psi = ||\mathcal{E}_\phi||_\psi$. Note that when we use a fitness function $f \neq t$ then we usually obtain the overall result of the evolutionary algorithm by computing $||\mathcal{E}_f||_t$ or $|\mathcal{E}_f|_t$ if we are confident about the elitism at least to the extent that we do not worry about substantial results getting lost along the way. In most cases we will assume that if f is close enough to t at least in the final generations, elitism with respect to f grants us quasi-elitism with respect t, i.e., if $f \approx t$ and $\mathcal{E}_f$ is elitist with respect to f, we assume that $||\mathcal{E}_f||_t \approx ||\mathcal{E}_f||_f$.

2.3 Example

We provide a running example accompanying these definitions.[4] For a target function, we choose two common benchmark functions from literature as they are implemented in the DEAP framework [2,12]. The first problem is based on the two-dimensional Schwefel function although we adjusted the target value space to fit comfortably within $[0; 1]$ (cf. Fig. 1a). We chose only two dimensions for ease of visualization. Higher-dimensional Schwefel is also covered in [8]. The Schwefel function is characterized by many valleys and hills of varying depth. The global optimum is at $X = Y \approx 420$. By contrast, our second example is the H1 function [14] that features one very distinct global optimum at $X = 8.6998, Y = 6.7665$. However, it feature very many little (hard to see) local optima throughout the whole surface. We took the classical H1 function, which is defined as a maximization problem and turned it upside down to produce a minimization problem (cf. Fig. 1b). For both target functions $t \in \{t_{Schwefel}, t_{H1}\}$ we construct the same evolutionary algorithm.

The search space is given as $\mathcal{X}_{Schwefel} = [-500; 500] \subseteq \mathbb{R}^2$ and $\mathcal{X}_{H1} = [-100; 100] \subseteq \mathbb{R}^2$ respectively. We initialize the search by generating X_0 from 25 random samples within the search space in both cases. The size of this population remains constant with application of the evolutionary step function e, which is constructed according to Eq. 1 with $A_1 = 0.3 \cdot |X|, A_2 = 0.1 \cdot |X|, A_3 = 0.1 \cdot |X|$.

[4] The code for all examples can be found at github.com/thomasgabor/isola-evolib.



(a) Normalized two-dimensional Schwefel (b) Inverse normalized H1

Fig. 1. Benchmark target functions used for the running example.

Let w be the range of a single dimensional value in the search space (i.e., $w_{Schwefel} = 1000, w_{H1} = 200$), then the mutation operator returns

$$mut((X, Y)) \in \{(X \oplus \delta, Y), (X, Y \oplus \delta) \mid \delta \in [-0.1w; 0.1w]\} \tag{4}$$

chosen random uniform where $\oplus$ only adds or subtracts as much of its second argument so that the resulting value remains within the search space. We further define the recombination operator so that its result is at random uniform picked from

$$rec((X, Y), (X', Y')) \in \{(X, Y), (X, Y'), (X', Y), (X', Y')\}. \tag{5}$$

Note that both operators include random cases where the operator does not do anything at, which does not harm the overall search and can be further counter-acted by increasing the respective amount of selected individuals for that operator. The migration operator just samples random uniform from the search space, returning $mig() \in \mathcal{X}$.

To illustrate the behavior of evolution, we ran independently initialized evolutionary processes for each problem 500 times each for 50 generations. Figure 2 shows all solution candidates found within a specific generation among *all* evolutionary processes. We can clearly trace how the search start random uniform and then focuses towards the global optima, sometimes getting stuck in local optima in the target value landscape (compare Fig. 1).

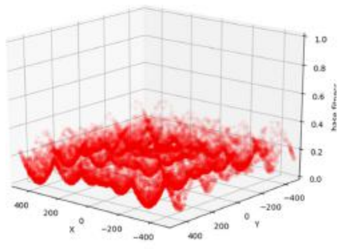
3 Approach

We apply the framework to give the definition of productive fitness. To present the full proof design we introduce and discuss Assumptions 1 and 2. We continue our example in Sect. 3.3.

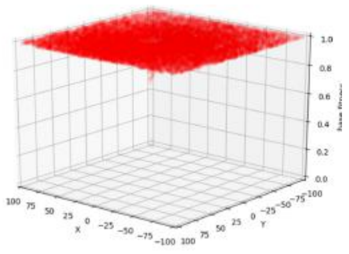
3.1 The Ideal Fitness

So far, we discussed some example definitions using the target function as fitness, $f = t$, and noted that it works (but not optimally). Obviously, having f correlate

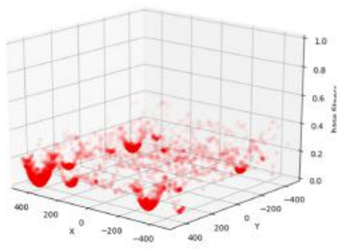
Taken from original publication: Thomas Gabor and Claudia Linnhoff-Popien. A formal model for reasoning about the ideal fitness in evolutionary processes. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2020



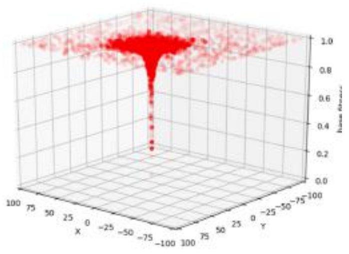
(a) Schwefel, generation 1



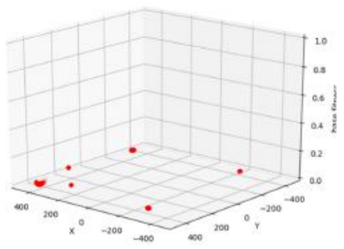
(b) H1, generation 1



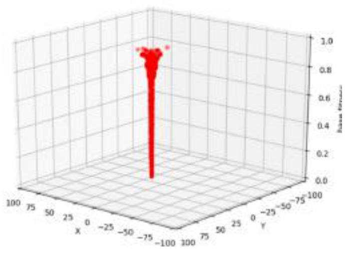
(c) Schwefel, generation 10



(d) H1, generation 10



(e) Schwefel, generation 50



(f) H1, generation 50

Fig. 2. Individuals from 500 independent runs of the evolutionary processes.

to some extend with t is a good thing if in the end we value our results with respect to t. However, it has long been known that augmenting the fitness with additional (meta-)information can greatly aid the optimization process in some cases. This fact is extensively discussed in literature [1,15] including previous works by the authors [7,8]. We sum the results up in the following observation:

Taken from original publication: Thomas Gabor and Claudia Linnhoff-Popien. A formal model for reasoning about the ideal fitness in evolutionary processes. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2020

Observation 1. *There exist evolutionary processes $\mathcal{E}_\phi = (\mathcal{X}, E_\phi, t, \langle X_i \rangle_{i \leq g})$ and $\mathcal{E}_t = (\mathcal{X}, E_t, t, \langle X'_i \rangle_{i \leq g})$ whose configurations only differ in the fitness function and there exist fitness functions $\phi \neq t$ so that $||\mathcal{E}_\phi||_t < ||\mathcal{E}_t||_t$.*

Observation 1 states that an evolutionary process can yield better results *with respect to t* by *not using t* directly but a somewhat approximate version of t given via ϕ, which includes additional information but likewise "waters down" the pure information of our original target. It is somewhat surprising that a deviation from the original target can yield an improvement. Commonly this phenomenon is explained by the *exploration/exploitation trade-off*: In an unknown solution landscape made up by t, we gain knowledge through evaluating solution candidates. When we have evaluated all solution candidates $x \in \mathcal{X}$, we simply need to compute $\arg\min_{x \in \mathcal{X}} t(x)$, which of course is infeasible for most practical search spaces. Giving limited time resources, we need to decide if we put additional effort into exploring more and new parts of the search space in hope of finding valuable solution candidates there or if we exploit the knowledge we have already gathered to further improve the solution candidates we already evaluated. This can be seen of a trade-off between large-scale search for exploration and small-scale search for exploitation.

Dealing with the exploration/exploitation trade-off certainly is one of the central tasks when implementing metaheuristic search and has been covered extensively in literature. Many of these approaches have been discovered bottom-up, often by analogy to biological or physical processes. Even though many similarities between approaches have been discovered, there does not exist a general framework for how to construct the right fitness function for a specific target function and evolutionary process.

Problem 1. *Given a target function t, what is the theoretically best fitness function ϕ^* for an evolutionary process $\mathcal{E}_{\phi^*}$ to optimize for t, i.e., optimize $||\mathcal{E}_{\phi^*}||_t$?*

We gave an answer to that question for the special case of standard evolutionary algorithms in [8]: We defined a measurement called *final productive fitness* and have sketched a proof that it represents the ideal fitness function for evolutionary algorithms. However, it is important to note that computing it a priori is infeasible. We approximated final productive fitness for an evolution a posteriori and provided empirical evidence that evolutionary algorithms are working better the better their fitness approximates final productive fitness.

In this paper, we formally introduce the necessary tools to provide the full proof of the ideal fitness for evolutionary algorithms. First, we need to re-iterate a few definitions of [8] in order to formally define final productive fitness.

Definition 9 (Descendants [8])**.** *Given an individual x in the population of generation i, $x \in X_i$, of an evolutionary process $\mathcal{E}$. All individuals $x' \in X_{i+1}$ so that x' resulted from x via a mutation operator, i.e., $x' = mut(x, r)$ for some $r \in \mathfrak{R}$, or a recombination operator with any other parent, i.e., there exists $y \in X_i$ so that $x' = rec(x, y, r)$ for some $r \in \mathfrak{R}$, are called* direct descendants *of x. Further given a series of populations $(X_i)_{0<i<g}$ we define the set of all descendants D_x as the transitive hull on all direct descendants of x.*

The main idea behind productive fitness is to measure an individual's effect on the optimization process. If the optimization process is stopping right now, i.e., if we are in the final generation g, then we can equate any individual's effect with its target function value. However, for any previous generations an individual's effect on the optimization corresponds to the best target function values that its descendants have achieved within the evolution.

Definition 10 (Productive Fitness [8]). *Given an individual x in the population of generation i, $x \in X_i$, of an evolutionary process $\mathcal{E}$. Let $D_x \subseteq \mathcal{X}$ be the set of all descendants from x. The* productive fitness after n generations *or* optimistic n-productive fitness *ϕ^+_n is the average achieved target value of x's descendants n generations later, written*

$$\phi^+_n(x) = \begin{cases} \mathrm{avg}_{x' \in D_x \cap X_{i+n}}\, t(x') & \textit{if } D_x \cap X_{i+n} \neq \emptyset \\ 1 & \textit{otherwise.} \end{cases} \tag{6}$$

Note that in case the individual x has no descendants in n generations, we set its productive fitness $\phi^+_n(x)$ to a worst case value of 1.

From [8] we repeat two major arguments against this definition:

– The use of avg as an aggregator over target values might be a bit pessimistic. By doing so, we penalize an individual's fitness if that individual bloats up the optimization with many low-value individuals. However, if it thereby also delivers at least one superior descendant, we should actually be fine with that when we only care about the end result. If such effects actually occur in practical scenarios is up to future work to discover. Empirical evidence discovered in [8] strongly argues in favor of using the average, which is why we repeat it in this definition. In the proof we will later derive a min-version from one of our assumptions.
– Assigning the value 1 in case the given individual has no further descendants in generation $i + n$ is a design choice. We might leave the productive fitness in this case undefined or at least assign a value outside the common range of target function values. We suggest that even without living descendants there might still be inherent value to having explored certain solution candidates (and having them clearly discarded for the ongoing process). Still, determining this incentive is up to future research.

Of course, productive fitness ϕ^+_n only measures the effect locally after a fixed amount of generations. For the effect for the whole evolution we can now easily define the notion of final productive fitness.

Definition 11 (Final Productive Fitness). *Given an individual x in the population of generation i, $x \in X_i$, of an evolutionary process $\mathcal{E}$ with g generations in total. The final productive fitness of x is the fitness of its descendants in the final generation, i.e.,*

$$\phi^\dagger(x) = \phi^+_{g-i}(x). \tag{7}$$

Taken from original publication: Thomas Gabor and Claudia Linnhoff-Popien. A formal model for reasoning about the ideal fitness in evolutionary processes. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2020

Described shortly, the final productive fitness of an individual x can be seen as an answer to the question: "How much did x contribute to the fitness of the individuals of the final population?" We claim that optimizing for that measurement results in the optimal evolutionary process (as considered in Problem 1).

Practically, of course, optimizing for that measurement is rather difficult (which in fact may be the entire reason it is the optimal fitness function): To make the completely right decision in generation $i = 1$, we would have to evaluate all possible future generations for each single individual being involved in any selection or altered in any way by evolutionary operators. Within a single generation, these are exponentially many possibilities, which of course grow exponentially with each generation. Still, in [8] we designed some approximation of final productive fitness that can at least be computed a posteriori for an already run evolution, giving some insight into the algorithm's workings. In this paper, we now provide the full design for a proof of final productive fitness's optimality, although there are still many risky new tools involved.

3.2 Proof Design

We hope that the notion of final productive fitness is intuitive enough so that it seems plausible how $\phi^\dagger$ might be the ideal fitness function ϕ^* for any evolutionary algorithm. However, evolutionary processes are highly stochastic entities and little framework exists to reason about their performance. We now first provide such a framework, although we resort to making some strong assumptions along the way.

Assumption 1 (Random Agnosticism)**.** *Random effects residing in selection functions and evolutionary operators exert the same general effects on the evolutionary function E regardless of the used fitness function.*

The main intention behind Assumption 1 is, of course, to exclude any concept of randomness from the proof design. We effectively assume that the distribution of outcomes (i.e., selected individuals or generated children) depending on random inputs does not depend on the fitness function. At first, this is a certainly outlandish and strong assumption, especially as it allows us to deduce quite strong properties. We just give a few reasons why it might be viable:

– Wherever random effects are used, they are usually designed to break clear fitness borders (for example when using a fuzzy cutoff vs. a discrete cutoff). In these cases, random effects overpower the effect of the fitness function so that (in the extreme case, consider random selection) the used fitness function has little impact on the outcome. If we flip the perspective around, different fitness functions then also have little difference for the outcome.
– Within the evolutionary function E, typically lots of random effects come together. Even if some of their distributions are altered by using a different fitness function, as long as they are not altered towards a specific result, the effect may still cancel out on the larger scale. Basically, we expect the outcome distribution of the whole evolutionary function E to approach the normal distribution irregardless of (un-systematic) mix-ups.

Taken from original publication: Thomas Gabor and Claudia Linnhoff-Popien. A formal model for reasoning about the ideal fitness in evolutionary processes. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2020

– From practical perspective, the shape of outcome distributions is rarely considered directly when constructing an evolutionary algorithm. That should usually indicate that not much effect can be observed there in most cases.

Eventually, all these reasons are flawed, of course. Otherwise, we would not have kept Assumption 1. Still, we feel that proofs built on Assumption 1 might have practical relevance for the time being.

It would be natural to follow up the effective elimination of randomness (coming from Assumption 1) by replacing all possibly random outcomes with the expected value of the distribution and treating all function as non-stochastic. However, the expected value is still computed from the distribution, so this would not make things much easier. Instead, we opt of the ideal outcome, which can be derived much easier, but might shift effects drastically: A recombination operator that performs so bad on average that it brings down the whole evolutionary process might now look like it gives rise to a very effective evolutionary process just because it has a very small chance of getting a really good result.

Assumption 2 (Based Optimism)**.** *For an evolutionary function E with a limited amount of possible outcomes, the best possible outcome is representative for its expected average result.*

We recognize that "limited" is not fully defined here. We suggest that future work looks into enumerability or local boundedness. For practical purposes, however, it is clear which of the classic operators are affected: Random initialization and migration can generate individuals across the whole search space. If we minimize over their possible outcomes, the whole algorithm reaches the global optimum in a single step. Mutation and recombination (with any kind of selection) on the other side are limited operators: Given certain individuals as input parameters, they will only navigate a limited range of options related to those individuals. Again the main argument for the plausibility of Assumption 2 is that the results usually approach normal distribution anyway and there is no real reason why they should act any differently given exactly the two fitness functions we are about to compare. However, given that we completely alter the rules of evolutionary algorithms with this one, it is definitely a bold assumption. Further note how we interpret the qualification "best" in Assumption 2: For a given evolutionary function E, its notion of "best" corresponds to its fitness function. So if we choose the best of two evolutionary processes with different fitness functions, we might actually choose two different points in the outcome distribution (depending on the fitness), which again is an immensely powerful tool based on a big assumption.

What Assumption 2 then provides is means to simplify Definition 10: Productive fitness is defined as the average fitness of all descendants. We can now use the *best* fitness of the descendants to compute the fitness measurement which

Taken from original publication: Thomas Gabor and Claudia Linnhoff-Popien. A formal model for reasoning about the ideal fitness in evolutionary processes. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2020

we will call *optimistic productive fitness.*[5] Note that implementing these assumptions has a great effect on the behavior of the evolutionary process. However, we do not claim that they leave the evolutionary process intact, we just claim that a clearly better fitness function remains the better fitness function even in the altered setting.

The tools provided by Assumptions 1 and 2 are rather novel and very powerful, so we are aware that any results based on them should be taken with a great amount of caution. However, in order to present these tools at work, we can use them to provide a proof that final productive fitness $\phi^\dagger$ is one answer for Problem 1.

Proof 1 (Problem 1)**.** *Let $\mathcal{E}^\dagger = \langle \mathcal{X}, E^\dagger, t, (X_i^\dagger)_{i<g} \rangle$ be an evolutionary process using optimistic final productive fitness $\phi^\dagger$. Let $\mathcal{E}^* = \langle \mathcal{X}, E^*, t, (X_i^*)_{i<g} \rangle$ be an evolutionary process using a different (possibly more ideal) fitness ϕ^*. According to the transformation discussed in Sect. 2.1, let both $\mathcal{E}^\dagger$ and $\mathcal{E}^*$ be elitist. Let $X_0^\dagger = X_0^*$. We assume that $t(||\mathcal{E}^\dagger||_t) > t(||\mathcal{E}^*||_t)$, i.e., because of elitism*

$$\min_{x \in X_g^\dagger} t(x) > \min_{x \in X_g^*} t(x). \tag{8}$$

From Eq. 8 it follows that there exists an individual $x \in X_g^*$ so that $x \notin X_g^\dagger$ and $t(x) < \min_{y \in X_g^\dagger} t(y)$. The better individual x could not have been introduced into the population of $\mathcal{E}^*$ by migration (or random initialization for that matter) as we could use Assumption 1 to just introduce x into $\mathcal{E}^\dagger$ then.

Then x needs to stem from an individual x' that is an ancestor of x, i.e., $x \in D_{x'}$, so that x' was selected for survival in $\mathcal{E}^*$ and not in $\mathcal{E}^\dagger$, which implies that $\phi^\dagger(x') > \phi^*(x')$. However, since x is a possible descendant for x', the computation of $\phi^\dagger(x')$ should have taken $t(x)$ into account,[6] *meaning that x' should have survived in $\mathcal{E}^\dagger$ because of elitism after all, which contradicts the previous assumption (Eq. 8).* □

3.3 Example

We now illustrate the notion of productive fitness for our running example. For each of the individuals generated in Sect. 2.3 we computed an a posteriori approximation for final productive fitness: Basically, we took the descendants that *have in fact been generated* during evolution as a representative subset of

[5] Note that the best possible choice for the average fitness of descendants is the minimum of the possible descendants' fitness values. When we are allowed to adjust the random choice for the best possible outcomes, worse-than-optimal children will not be born. This changes the game: Our ideal choice from the vast space of random possibilities now yields at most one (i.e. the best possible) descendant per individual per generation.

[6] Note that $t(x)$ cannot be compensated by other descendants of x' with possibly bad objective fitness since we assumed optimistic final productive fitness following Assumption 2.

the descendants that *could have been generated*. This allows us to compute a value for $\phi^\dagger$ for an already finished evolution.[7]

Note that the notion of final productive fitness is most powerful in the beginning of the evolutionary process, when it carries information about the whole evolution to come. Figures 3a and 3b provide a clear situation how final productive fitness is a better fitness function than the target function:

- The final productive fitness landscape has fewer valleys as its local optima correspond to individuals that remained in the final generation of some evolutionary process. This makes the landscape less deceptive and individuals are more clearly guided towards at least somewhat good results.
- The basins around the optima are wider, again making the local optimization towards the final result more clear.
- The differences between the global optimum and other local optima are more pronounced, giving an edge to the global optimum.

As discussed, if we could use final productive fitness during evolution, it would allow for better results. However, approximations of various quality may exist for specific problems or problem instances [8].

In Figs. 3c and 3d we can see how the final productive fitness landscapes deteriorates with the progressing evolution. As we can see from the red dots, evolution has focused on certain areas of the solution landscape, leaving wide areas without a meaningful final productive fitness to be computed. This effect is even more prominent in Figs. 3e and 3f, where individuals that are still randomly generated in certain areas die out rather quickly, leaving them with a productive fitness of 1. Note that productive fitness cannot meaningfully be computed for the last generation so we deliberately choose to show generation 49 last here.

Note how Fig. 3 also illustrates the usage of different evolutionary operators: For the Schwefel function, many individuals have a good final productive fitness when the evolution starts. That means they have direct descendants who manage to achieve nearly optimal target values. By contrast, H1 shows no individuals with good productive fitness in the beginning, meaning that the final results were mostly discovered via the migration operator *mig* as that is not traced by productive fitness.[8]

4 Related Work

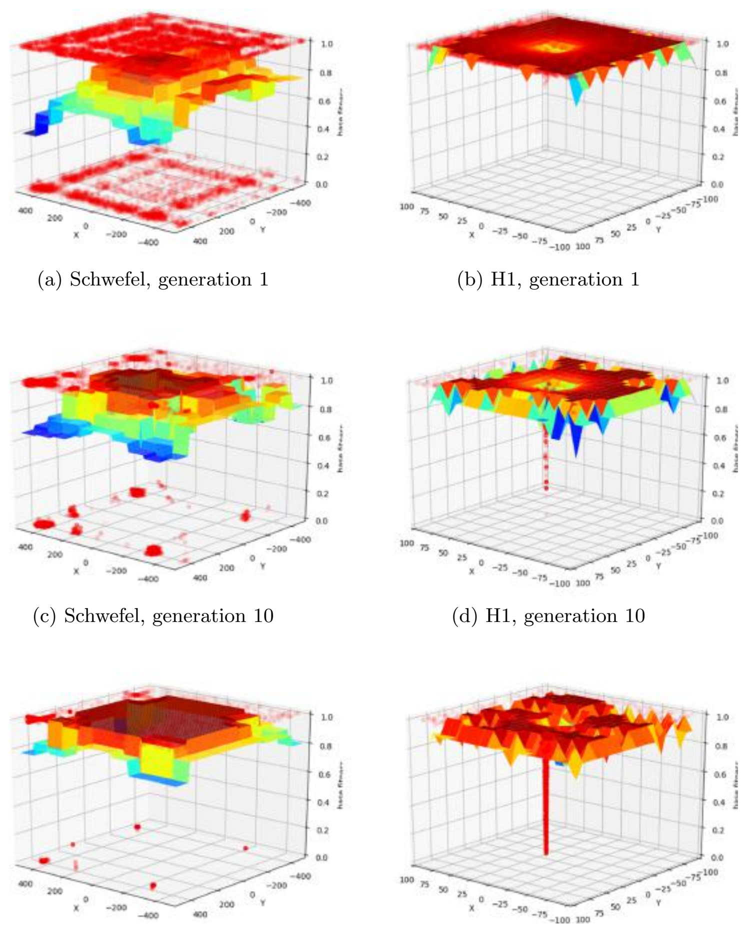
We first introduced the gist of the formal framework for evolutionary processes as well as the notion of productive fitness in [8]. In this paper, we provide and discuss the full, substantially extended framework and introduce the assumptions and tools a proof design for productive fitness's validity can be built with.

Theoretical work on evolutionary algorithms has been traditionally focused on the complexity of the search process (on rather simple search problems) or

[7] For details on how this is done, please refer to [8].

[8] Migrants are generated randomly and are thus not ascribed to be any individual's descendant. How to include migrants in productive fitness is left for future work.

Taken from original publication: Thomas Gabor and Claudia Linnhoff-Popien. A formal model for reasoning about the ideal fitness in evolutionary processes. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2020



(a) Schwefel, generation 1 (b) H1, generation 1

(c) Schwefel, generation 10 (d) H1, generation 10

(e) Schwefel, generation 49 (f) H1, generation 49

Fig. 3. Individuals from 500 independent runs of the evolutionary processes plotted with their a posteriori approximated final productive fitness. The surface represents the same data set as the scatter points, where each tile has the Z value equal to the average Z value of all the points within it.

Taken from original publication: Thomas Gabor and Claudia Linnhoff-Popien. A formal model for reasoning about the ideal fitness in evolutionary processes. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2020

the performance of various types and variants of algorithms in general. We point to [4–6] for a few selective examples without any attempt at giving a full overview over this old and comprehensive field of research. By contrast, we fully work out the difference between a target function that is given from the outside world and a fitness function that (potentially) emerges implicitly throughout the process of evolution. As this concept in itself is rather novel, the constructs supporting it have been freshly developed as well (and are still in their infancy).

It should be pointed out that there probably exists a connection from the assumptions and approximations we make to a complexity-based analysis of evolution, as these tools allow us to rule out exponentially many options and thus bring the respective computation to a feasible level.[9]

Various meta-measurements of fitness in evolutionary algorithms have been designed. We would like to point out *effective fitness* [13], which describes the fitness threshold under which individuals can manage to increase their dominance in the population. This usually is a harsher border than *reproductive fitness* [11], which is the probability of an individual to successfully produce offspring. Both follow a similar line of thought of measuring what fitness an individual needs to have for certain effects to occur, but none suggest using the meta-measurement as a fitness value itself.

5 Conclusion

We have introduced and discussed a formal description of evolutionary processes that summarizes various kinds of evolutionary algorithms and other optimization techniques. Based on that framework, we defined the notion of productive fitness as it is defined in [8], where an argument was sketched why it might be the ideal fitness function. In this paper, we introduced the tools necessary to implement the proof, discussed their validity and thus gave the full proof design. We argue that while the approach is somewhat bold, the assumptions made could be useful for similar arguments about evolutionary processes and hope the perspective on fitness functions given here will open up new ways to reason about highly dynamic and uncertain processes, especially evolution.

We pointed out future work where we encountered open questions. We consider the connection suggested to traditional runtime analysis of evolutionary algorithms and subsequently to the No Free Lunch theorem [10] and how it related to the cases of having and using as well as finding and approximating the ideal fitness function to be especially promising. In addition, we suggest that it might be of particular relevance to also expand the scope of the framework beyond evolutionary algorithms; even the proof design might be adapted to not only work for fitness used by evolutionary operators but for example to deliver the ideal reward function for reinforcement learning [3,9].

[9] As no computational limit on biological evolution, e.g., has been recognized it could be an interesting endeavor to use the framework presented in this paper to translate arguments from runtime analysis of evolutionary algorithms back to a more general concept of evolution.

Taken from original publication: Thomas Gabor and Claudia Linnhoff-Popien. A formal model for reasoning about the ideal fitness in evolutionary processes. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2020

References

1. Brown, G., Wyatt, J., Harris, R., Yao, X.: Diversity creation methods: a survey and categorisation. Inf. Fusion **6**(1), 5–20 (2005)
2. DEAP Project: Benchmarks (2020). https://deap.readthedocs.io/en/master/api/benchmarks.html. Accessed June 1 2020
3. Dewey, D.: Reinforcement learning and the reward engineering principle. In: 2014 AAAI Spring Symposium Series (2014)
4. Doerr, B., Happ, E., Klein, C.: Crossover can provably be useful in evolutionary computation. Theoret. Comput. Sci. **425**, 17–33 (2012)
5. Droste, S., Jansen, T., Wegener, I.: On the analysis of the (1+1) evolutionary algorithm. Theoret. Comput. Sci. **276**(1–2), 51–81 (2002)
6. Friedrich, T., Oliveto, P.S., Sudholt, D., Witt, C.: Analysis of diversity-preserving mechanisms for global exploration. Evol. Comp. **17**(4), 455–476 (2009)
7. Gabor, T., Belzner, L., Linnhoff-Popien, C.: Inheritance-based diversity measures for explicit convergence control in evolutionary algorithms. In: Genetic and Evolutionary Computation Conference, pp. 841–848 (2018)
8. Gabor, T., Phan, T., Linnhoff-Popien, C.: Productive fitness in diversity-aware evolutionary algorithms (2021). (submitted)
9. Hadfield-Menell, D., Milli, S., Abbeel, P., Russell, S.J., Dragan, A.: Inverse reward design. In: Advances in Neural Information Processing Systems, pp. 6765–6774 (2017)
10. Ho, Y.C., Pepyne, D.L.: Simple explanation of the no free lunch theorem of optimization. In: 40th IEEE Conference on Decision and Control (Cat. No. 01CH37228), vol. 5, pp. 4409–4414. IEEE (2001)
11. Hu, T., Banzhaf, W.: Evolvability and speed of evolutionary algorithms in light of recent developments in biology. J. Artif. Evol. Appl. **2010**, 1 (2010)
12. Rainville, D., Fortin, F.A., Gardner, M.A., Parizeau, M., Gagné, C., et al.: Deap: a python framework for evolutionary algorithms. In: Conference Companion on Genetic and Evolutionary Computation, pp. 85–92. ACM (2012)
13. Stephens, C.R.: "Effective" fitness landscapes for evolutionary systems. In: 1999 Congress on Evolutionary Computation (CEC 1999), vol. 1, pp. 703–714. IEEE (1999)
14. Van Soest, A.K., Casius, L.R.: The merits of a parallel genetic algorithm in solving hard optimization problems. J. Biomech. Eng. **125**(1), 141–146 (2003)
15. Wineberg, M., Oppacher, F.: The underlying similarity of diversity measures used in evolutionary computation. In: Cantú-Paz, E., et al. (eds.) GECCO 2003. LNCS, vol. 2724, pp. 1493–1504. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-45110-2_21

Taken from original publication: Thomas Gabor, Steffen Illium, Andy Mattausch, Lenz Belzner, and Claudia Linnhoff-Popien. Self-replication in neural networks. In *Artificial Life Conference Proceedings*, pages 424–431. MIT Press, 2019

Self-Replication in Neural Networks

Thomas Gabor[1], Steffen Illium[1], Andy Mattausch[1], Lenz Belzner[2], Claudia Linnhoff-Popien[1]

[1]LMU Munich
[2]MaibornWolff
thomas.gabor@ifi.lmu.de

Abstract

The foundation of biological structures is self-replication. Neural networks are the prime structure used for the emergent construction of complex behavior in computers. We analyze how various network types lend themselves to self-replication. We argue that backpropagation is the natural way to navigate the space of network weights and show how it allows non-trivial self-replicators to arise naturally. We then extend the setting to construct an artificial chemistry environment of several neural networks.

Introduction

Dawkins (1976) stressed the importance of self-replication to the origin of life. He argued that proto-RNA was able to copy its molecule structure within a soup of randomly interacting elements. This allowed it to reach a stability in concentration that could not be maintained by any other kind of structure. Eventually, life evolved more or less as an elaborate means to maintain the copying of structural information.

Since the early days of computing, the recreation of biological structures has been a target of research, starting from the early formulation of an evolutionary process by Turing (1950) and including famous examples like Box (1957), Conway (1970) or Dorigo and Di Caro (1999). Also see the overviews given by Koza (1994) or Bäck et al. (1997). Although conceived very early as well (Rosenblatt (1958) and Minsky and Papert (1972)), neural networks have only recently found broad practical application for advanced tasks like image recognition (Krizhevsky et al. (2012)), speech recognition (Hinton et al. (2012)) or strategic game playing (Silver et al. (2017)). The variety of uses shows that neural networks are a powerful tool of abstraction for various domains. However, in all these cases neural networks are used with a certain intend, i.e., equipped with a goal function. Through backpropagation, the distance of the network's output to the goal function can be systematically minimized.

The wide variety of application domains shows the power of neural networks as a functional abstraction. For other functional abstractions, such as expressions in the λ-calculus (Church (1932)) or a variety of assembler-like instruction sets and automata (Dittrich et al. (2001)), it is known that, when a population of random instances of said functional abstractions are set up and allowed to interact, self-replicators arise naturally (see Fontana and Buss (1996) or Dittrich and Banzhaf (1998), respectively). For neural networks, Chang and Lipson (2018) have shown that self-application may lead to the formation of a self-replicating structure, albeit a rather trivial instance of one. In this paper, we (a) repeat these results for a broader range of neural network architectures and (b) extend the interaction model by the notion of self-training, which yields lots of non-trivial fixpoints. This allows us to (c) construct an artificial chemistry setup using neural networks as individuals that (under certain circumstances, of course) reliably produces a variety of non-trivial self-replicators.

Foundations

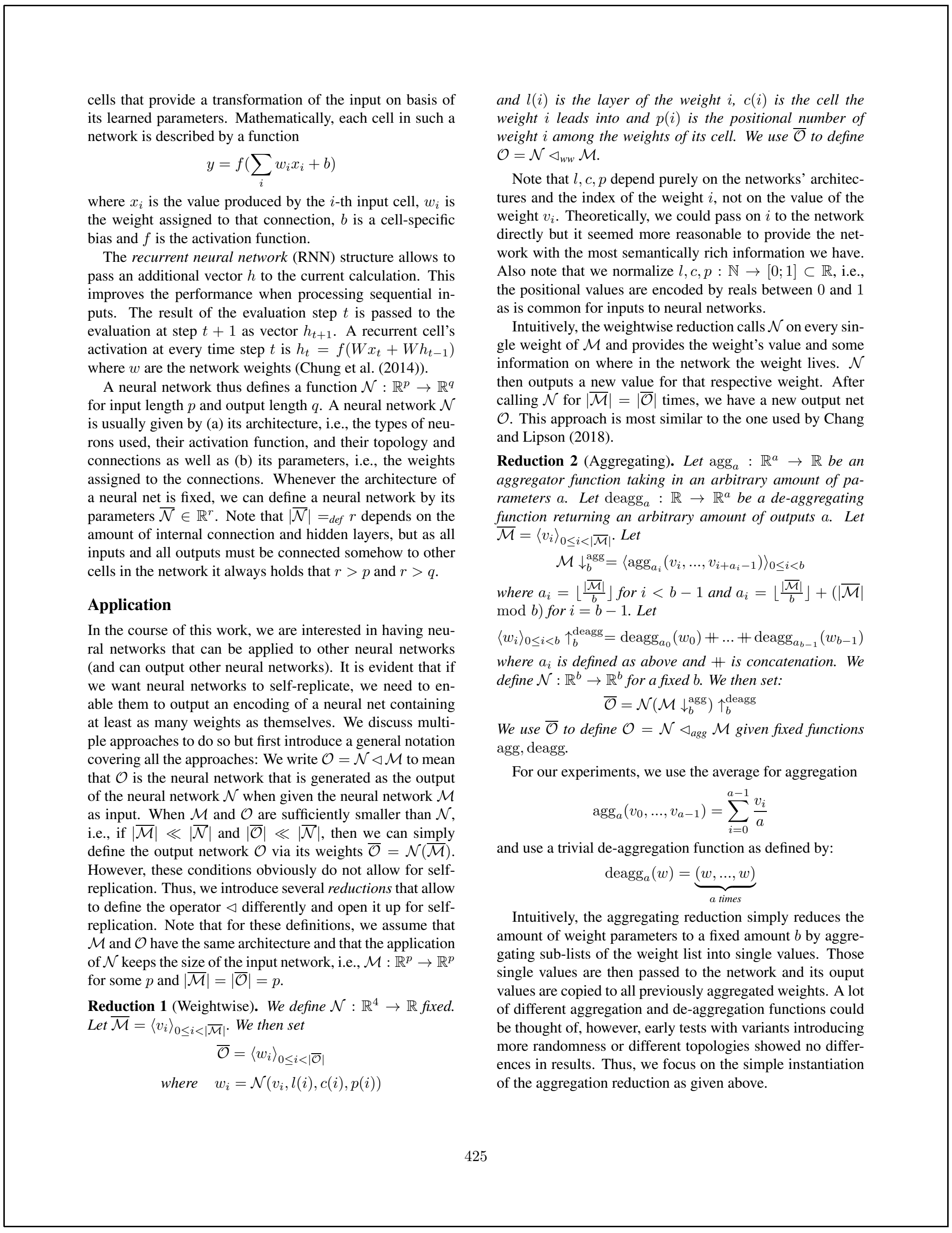
We provide a brief introduction to how neural networks function, then we proceed to discuss how to apply neural networks to other neural networks and how to train neural networks using other neural networks.

Basics

Neural networks are most commonly made from layers of neurons that are connected to the next layers of neurons and so on. As there are many kinds of neurons (fully connected, recurrent, convolutional) there are also many kinds of layers. Variations of this scheme go up to well established structures within such layers, consisting not only of single functional cells (LSTM, attention mechanism). What they have in common is the base functionality of accepting values (in form of a matrix or vector), application of weights or bias (a matrix of a similar shape, also known as the network's parameters), followed by a specific activation function (linear, sigmoid, relu, tanh, e.g.) that transforms the outputs. Note that while neural networks originated as a model of biological neurons, they cannot accurately fulfill that role anymore and instead serve as general function approximators.

The most basic form of a feed-forward network is a single-layer perceptron, consisting of many fully connected

Taken from original publication: Thomas Gabor, Steffen Illium, Andy Mattausch, Lenz Belzner, and Claudia Linnhoff-Popien. Self-replication in neural networks. In *Artificial Life Conference Proceedings*, pages 424–431. MIT Press, 2019



cells that provide a transformation of the input on basis of its learned parameters. Mathematically, each cell in such a network is described by a function

$$y = f(\sum_i w_i x_i + b)$$

where x_i is the value produced by the i-th input cell, w_i is the weight assigned to that connection, b is a cell-specific bias and f is the activation function.

The *recurrent neural network* (RNN) structure allows to pass an additional vector h to the current calculation. This improves the performance when processing sequential inputs. The result of the evaluation step t is passed to the evaluation at step $t+1$ as vector h_{t+1}. A recurrent cell's activation at every time step t is $h_t = f(Wx_t + Wh_{t-1})$ where w are the network weights (Chung et al. (2014)).

A neural network thus defines a function $\mathcal{N} : \mathbb{R}^p \to \mathbb{R}^q$ for input length p and output length q. A neural network $\mathcal{N}$ is usually given by (a) its architecture, i.e., the types of neurons used, their activation function, and their topology and connections as well as (b) its parameters, i.e., the weights assigned to the connections. Whenever the architecture of a neural net is fixed, we can define a neural network by its parameters $\overline{\mathcal{N}} \in \mathbb{R}^r$. Note that $|\overline{\mathcal{N}}| =_{def} r$ depends on the amount of internal connection and hidden layers, but as all inputs and all outputs must be connected somehow to other cells in the network it always holds that $r > p$ and $r > q$.

Application

In the course of this work, we are interested in having neural networks that can be applied to other neural networks (and can output other neural networks). It is evident that if we want neural networks to self-replicate, we need to enable them to output an encoding of a neural net containing at least as many weights as themselves. We discuss multiple approaches to do so but first introduce a general notation covering all the approaches: We write $\mathcal{O} = \mathcal{N} \lhd \mathcal{M}$ to mean that $\mathcal{O}$ is the neural network that is generated as the output of the neural network $\mathcal{N}$ when given the neural network $\mathcal{M}$ as input. When $\mathcal{M}$ and $\mathcal{O}$ are sufficiently smaller than $\mathcal{N}$, i.e., if $|\overline{\mathcal{M}}| \ll |\overline{\mathcal{N}}|$ and $|\overline{\mathcal{O}}| \ll |\overline{\mathcal{N}}|$, then we can simply define the output network $\mathcal{O}$ via its weights $\overline{\mathcal{O}} = \mathcal{N}(\overline{\mathcal{M}})$. However, these conditions obviously do not allow for self-replication. Thus, we introduce several *reductions* that allow to define the operator $\lhd$ differently and open it up for self-replication. Note that for these definitions, we assume that $\mathcal{M}$ and $\mathcal{O}$ have the same architecture and that the application of $\mathcal{N}$ keeps the size of the input network, i.e., $\mathcal{M} : \mathbb{R}^p \to \mathbb{R}^p$ for some p and $|\overline{\mathcal{M}}| = |\overline{\mathcal{O}}| = p$.

Reduction 1 (Weightwise)**.** *We define $\mathcal{N} : \mathbb{R}^4 \to \mathbb{R}$ fixed. Let $\overline{\mathcal{M}} = \langle v_i \rangle_{0 \leq i < |\overline{\mathcal{M}}|}$. We then set*

$$\overline{\mathcal{O}} = \langle w_i \rangle_{0 \leq i < |\overline{\mathcal{O}}|}$$

$$\textit{where} \quad w_i = \mathcal{N}(v_i, l(i), c(i), p(i))$$

and $l(i)$ is the layer of the weight i, $c(i)$ is the cell the weight i leads into and $p(i)$ is the positional number of weight i among the weights of its cell. We use $\overline{\mathcal{O}}$ to define $\mathcal{O} = \mathcal{N} \lhd_{ww} \mathcal{M}$.

Note that l, c, p depend purely on the networks' architectures and the index of the weight i, not on the value of the weight v_i. Theoretically, we could pass on i to the network directly but it seemed more reasonable to provide the network with the most semantically rich information we have. Also note that we normalize $l, c, p : \mathbb{N} \to [0; 1] \subset \mathbb{R}$, i.e., the positional values are encoded by reals between 0 and 1 as is common for inputs to neural networks.

Intuitively, the weightwise reduction calls $\mathcal{N}$ on every single weight of $\mathcal{M}$ and provides the weight's value and some information on where in the network the weight lives. $\mathcal{N}$ then outputs a new value for that respective weight. After calling $\mathcal{N}$ for $|\overline{\mathcal{M}}| = |\overline{\mathcal{O}}|$ times, we have a new output net $\mathcal{O}$. This approach is most similar to the one used by Chang and Lipson (2018).

Reduction 2 (Aggregating)**.** *Let* $\text{agg}_a : \mathbb{R}^a \to \mathbb{R}$ *be an aggregator function taking in an arbitrary amount of parameters a. Let* $\text{deagg}_a : \mathbb{R} \to \mathbb{R}^a$ *be a de-aggregating function returning an arbitrary amount of outputs a. Let $\overline{\mathcal{M}} = \langle v_i \rangle_{0 \leq i < |\overline{\mathcal{M}}|}$. Let*

$$\mathcal{M} \downarrow_b^{\text{agg}} = \langle \text{agg}_{a_i}(v_i, ..., v_{i+a_i-1}) \rangle_{0 \leq i < b}$$

where $a_i = \lfloor \frac{|\overline{\mathcal{M}}|}{b} \rfloor$ for $i < b-1$ and $a_i = \lfloor \frac{|\overline{\mathcal{M}}|}{b} \rfloor + (|\overline{\mathcal{M}}| \bmod b)$ for $i = b-1$. Let

$$\langle w_i \rangle_{0 \leq i < b} \uparrow_b^{\text{deagg}} = \text{deagg}_{a_0}(w_0) \mathbin{+\!\!+} ... \mathbin{+\!\!+} \text{deagg}_{a_{b-1}}(w_{b-1})$$

where a_i is defined as above and $\mathbin{+\!\!+}$ is concatenation. We define $\mathcal{N} : \mathbb{R}^b \to \mathbb{R}^b$ for a fixed b. We then set:

$$\overline{\mathcal{O}} = \mathcal{N}(\mathcal{M} \downarrow_b^{\text{agg}}) \uparrow_b^{\text{deagg}}$$

We use $\overline{\mathcal{O}}$ to define $\mathcal{O} = \mathcal{N} \lhd_{agg} \mathcal{M}$ given fixed functions agg, deagg.

For our experiments, we use the average for aggregation

$$\text{agg}_a(v_0, ..., v_{a-1}) = \sum_{i=0}^{a-1} \frac{v_i}{a}$$

and use a trivial de-aggregation function as defined by:

$$\text{deagg}_a(w) = \underbrace{(w, ..., w)}_{a \text{ times}}$$

Intuitively, the aggregating reduction simply reduces the amount of weight parameters to a fixed amount b by aggregating sub-lists of the weight list into single values. Those single values are then passed to the network and its ouput values are copied to all previously aggregated weights. A lot of different aggregation and de-aggregation functions could be thought of, however, early tests with variants introducing more randomness or different topologies showed no differences in results. Thus, we focus on the simple instantiation of the aggregation reduction as given above.

Taken from original publication: Thomas Gabor, Steffen Illium, Andy Mattausch, Lenz Belzner, and Claudia Linnhoff-Popien. Self-replication in neural networks. In *Artificial Life Conference Proceedings*, pages 424–431. MIT Press, 2019

Reduction 3 (RNN)**.** *We define $\mathcal{N} : \mathbb{R} \times \mathbb{R}^H \to \mathbb{R} \times \mathbb{R}^H$ as a recurrent neural network with a hidden state $h \in \mathbb{R}^H$ for some $H \in \mathbb{N}$. Let $\overline{\mathcal{M}} = \langle v_i \rangle_{0 \leq i < |\overline{\mathcal{M}}|}$. We then set*

$$\overline{\mathcal{O}} = \langle w_i \rangle_{0 \leq i < |\overline{\mathcal{O}}|}$$

where w_i is given via

$$\mathcal{N}(v_i, h_i) = (w_i, h_{i+1})$$

where $h_0 = \mathbf{0}$. We use $\overline{\mathcal{O}}$ to define $\mathcal{O} = \mathcal{N} \lhd_{rnn} \mathcal{M}$.

Since recurrent neural networks are able to process input sequences of arbitrary length, the RNN reduction technically just needs to define $\mathcal{N}$ as a recurrent neural network and simply apply it to the weights of another network. Even though this reduction appears most simple and natural, the explosion of gradients within larger recurrent neural networks means that they are very prone to diverge to very large output values if not sufficiently controlled. We reckon that an extension to recurrent neural networks (making them accessible to self-replication) should be possible, however, since vanilla recurrent neural networks are not so fit for self-replication, we refer this extension to future work.

We can use these several types of reduction to build a mathematical model of self-replication in neural networks.

Definition 1 (Self-Application)**.** *Given a neural network $\mathcal{N}$. Let $\lhd$ be a suitable reduction. We call the neural network $\mathcal{N}' = \mathcal{N} \lhd \mathcal{N}$ the self-application of $\mathcal{N}$.*

Definition 2 (Fixpoint, Self-Replication)**.** *Given a neural network $\mathcal{N}$. Let $\lhd$ be a suitable reduction. We call $\mathcal{N}$ a fixpoint with respect to $\lhd$ iff $\mathcal{N} = \mathcal{N} \lhd \mathcal{N}$, i.e., iff $\mathcal{N}$ is its own self-application. We also say that $\mathcal{N}$ is able to self-replicate.*

Since network weights are real-valued and are the result of many computations, checking for the equality $\mathcal{N} = \mathcal{N} \lhd \mathcal{N}$ is not entirely trivial. We thus relax the fixpoint property a bit.

Definition 3 (ε-Fixpoint)**.** *Given a neural network $\mathcal{N}$ with weights $\overline{\mathcal{N}} = \langle v_i \rangle_{0 \leq i < |\overline{\mathcal{N}}|}$. Let $\lhd$ be a suitable reduction. Let $\varepsilon \in \mathbb{R}$ be the error margin of the fixpoint property. Let $\mathcal{N}' = \mathcal{N} \lhd \mathcal{N}$ be the self-application of $\mathcal{N}$ with weights $\overline{\mathcal{N}'} = \langle w_i \rangle_{0 \leq i < |\overline{\mathcal{N}'}|}$. We call $\mathcal{N}$ an ε-fixpoint or a fixpoint up to ε iff for all i it holds that $|w_i - v_i| < \varepsilon$.*

Training

As stated above, neural networks are commonly used in conjunction with backpropagation to adjust their weights to a desired configuration. We assume that we have a set of input vectors $\mathbf{x}_0, ..., \mathbf{x}_n$ and a corresponding set of desired output vectors $\mathbf{y}_0, .., \mathbf{y}_n$. We want our neural network $\mathcal{N}$ to represent the relation between these sets. The loss for a single sample $(\mathbf{x}_i, \mathbf{y}_i)$ is defined as $|\mathcal{N}(\mathbf{x}_i) - \mathbf{y}_i|$. Minimizing the loss of a neural network is called *training*. We use the SGD optimizer to apply gradient updates or rather weight changes to minimize the loss for given a given sample $(\mathbf{x}_i, \mathbf{y}_i)$, which results in an updated network $\mathcal{N}' = \mathcal{N} \leftsquigarrow (\mathbf{x}_i, \mathbf{y}_i)$. We call $\leftsquigarrow$ the training operator. For sets of sample points $\mathbf{x} = \mathbf{x}_0, ..., \mathbf{x}_n$ and $\mathbf{y} = \mathbf{y}_0, ..., \mathbf{y}_n$, we also write $\mathcal{N} \leftsquigarrow \mathbf{x}, \mathbf{y}$ as shorthand for $\mathcal{N} \leftsquigarrow (\mathbf{x}_0, \mathbf{y}_0) \leftsquigarrow ... \leftsquigarrow (\mathbf{x}_n, \mathbf{y}_n)$.

We argue that training neural networks is another natural way of evolving them (as is application). Thus, we also want to train a neural network with other neural networks as input and output data. Of course, we again need to use reduction on said other neural networks. In short we write:

Reduction 4 (Weightwise Training)**.** *Given neural networks $\mathcal{M}, \mathcal{N}$ with $\overline{\mathcal{M}} = \langle v_i \rangle_{0 \leq i \leq n}$ for some n. We write $\mathcal{N}' = \mathcal{N} \leftsquigarrow_{ww} \mathcal{M}$ iff*

$$\mathcal{N} = \mathcal{N} \leftsquigarrow \langle (v_i, l(i), c(i), p(i)) \rangle_{0 \leq i \leq n}, \langle (v_i) \rangle_{0 \leq i \leq n}$$

where l, c, p are defined as in Reduction 1.

Reduction 5 (Aggregating Training)**.** *Given neural networks $\mathcal{M}, \mathcal{N}$. Given a suitable aggregator function* agg *and aggregated size b. We write $\mathcal{N}' = \mathcal{N} \leftsquigarrow_{agg} \mathcal{M}$ iff*

$$\mathcal{N}' = \mathcal{N} \leftsquigarrow (\mathcal{M} \downarrow_b^{\text{agg}}, \mathcal{M} \downarrow_b^{\text{agg}})$$

where the $\downarrow$ operation is defined as in Reduction 2.

Reduction 6 (Recurrent Training)**.** *Given neural networks $\mathcal{M}, \mathcal{N}$. We write $\mathcal{N}' = \mathcal{N} \leftsquigarrow_{rnn} \mathcal{M}$ iff*

$$\mathcal{N}' = \mathcal{N} \leftsquigarrow (\overline{\mathcal{M}}, \overline{\mathcal{M}})$$

where $\mathcal{N}$ is trained on a sequence $\overline{\mathcal{M}}$ by being applied one by one recurrently.

Intuitively, these training reductions transform the input net $\mathcal{M}$ to a smaller representation (as do the application reductions, cf. Reductions 1–3) and then train the network $\mathcal{N}$ to accurately reproduce that representation.

Note that usually, when training a neural network, we derive training samples from a large data set or generate them automatically. However, we can use these training reductions to define the notion of self-training:

Definition 4 (Self-Training)**.** *Given a neural network $\mathcal{N}$. Let $\leftsquigarrow$ be a suitable training reduction. We call the network $\mathcal{N}' = \mathcal{N} \leftsquigarrow \mathcal{N}$ the result of self-training $\mathcal{N}$.*

We can apply self-training for many consecutive steps, however, in contrast to usual training in neural networks, the samples made available for training only depend on the network's own weights and introduce no randomness or additional coverage of the search space beyond their own (mostly pre-determined) evolution via self-training.

Experiments

We define three types of experiments, which test the two distinct approaches to self-replication based on application of neural networks to other neural networks and training using backpropagation on self-generated limited training points, respectively. Lastly, we show a strong connection between both approaches.

Note that for the sake of simplicity, we fixed all network architectures in the following experiments to only include

426

Taken from original publication: Thomas Gabor, Steffen Illium, Andy Mattausch, Lenz Belzner, and Claudia Linnhoff-Popien. Self-replication in neural networks. In *Artificial Life Conference Proceedings*, pages 424–431. MIT Press, 2019

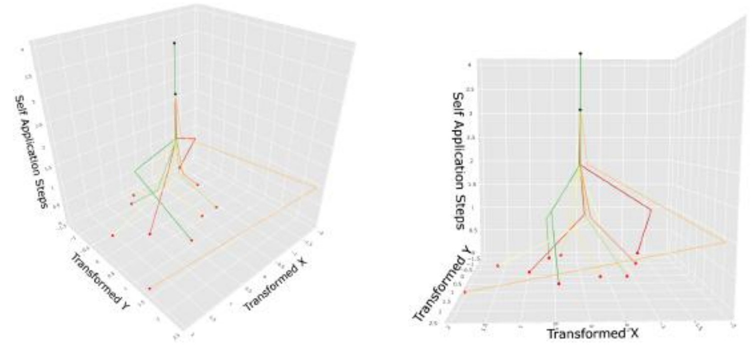


Figure 1: 10 independent runs of **self-application** with respect to the aggregating reduction $\lhd_{agg}$. 10 neural networks $\mathcal{N} : \mathbb{R}^4 \rightarrow \mathbb{R}^4$ with two hidden layers with two cells each were initialized randomly and then subjected to 4 self-applications each. The figure shows two perspectives on the same three-dimensional graph. The 20 weights in total per network were visualized in a two-dimensional space based on the transformed bases X and Y derived via PCA. All networks converge on (X=0, Y=0), which corresponds to the weight vector **0**.

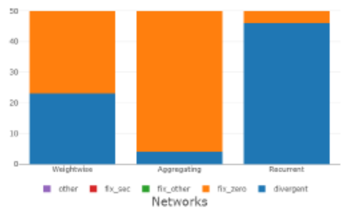


Figure 2: 50 independent runs of **self-application** each with respect to all three different types of reduction. We show an analysis of the networks (which were initialized randomly) after 100 steps of self-application.

two hidden layers with two cells each. Although evaluations were run with various activation functions, all plots show linear activation since we observed no qualitative difference between various activations. Similarly, bias was set to 0 in all plotted instances.

Self-Application

When subjecting a randomly initialized neural network $\mathcal{N}$ to repeated self-application with respect to the weightwise reduction $\lhd_{ww}$, the weight vector $\overline{\mathcal{N}}$ tends to converge to the all-zero vector $\mathbf{0} = \langle 0 \rangle_{|\overline{\mathcal{M}}|}$. This was already indicated by Chang and Lipson (2018) for a very similar reduction approach. This effect probably stems from a phenomenon observed by Schoenholz et al. (2017): Randomly initialized neural networks tend to map their inputs to output values closer to **0**. Figure 1 shows that the same effect also occurs for the aggregating reduction $\lhd_{agg}$. It shows the journey of several neural networks through the space of weight vectors.[1] Very few steps of self-application suffice to draw all neural networks to the coordinates (X=0, Y=0), which in fact correspond to the weight vector **0**.

The same plot for the weightwise reduction $\lhd_{ww}$ looks rather similar. Figure 2 shows the resulting networks after several steps of self-application. Here, we discern five observations: A neural network $\mathcal{N}$ is (a) *divergent* iff at any point in time any of its weights assumed the value ∞ or $-\infty$. Once this has happened, there is no returning from it. If the network assumes (b) the ε-fixpoint given by the weight vector **0**, i.e., all its weights are sufficiently close to 0, we call the network a *zero fixpoint*. Note that for all experiments we set $\varepsilon = 10^{-5}$. If the network's weights resemble (c) any other ε-fixpoint we call it a non-zero, non-trivial or simply *other fixpoint*. At this stage, we also checked for (d) *second-order fixpoints*, i.e., networks $\mathcal{N}$ fulfilling the weaker property $\mathcal{N} = \mathcal{N} \lhd \mathcal{N} \lhd \mathcal{N}$. However, we never found any such networks. Anything else falls into the category (e) *other*.

[1]To be able to plot highly-dimensional weight vectors on paper, we derive the two principle components of the observed weight vectors using standard principle component analysis (PCA) and plot the weight vector as a point in that two-dimensional space. We use this technique for all such figures.

Taken from original publication: Thomas Gabor, Steffen Illium, Andy Mattausch, Lenz Belzner, and Claudia Linnhoff-Popien. Self-replication in neural networks. In *Artificial Life Conference Proceedings*, pages 424–431. MIT Press, 2019

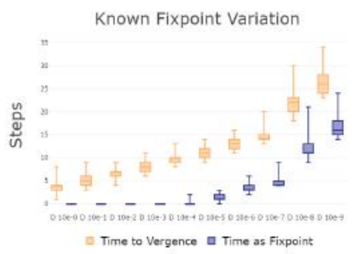


Figure 3: The robustness of a known 10^{-5}-fixpoint with respect to the weightwise reduction $\lhd_{ww}$. The x-axis shows the range of noise that the known fixpoint's weights were subjected to. The y-axis shows for how many steps of self-application the network was still regarded as a 10^{-5}-fixpoint (purple) and after how many steps of self-application the network was regarded as diverged (orange).

Note that Figure 2 shows that no non-zero fixpoints are found for any reduction and that recurrent neural nets are most prone to diverge during repeated self-application.

We also checked for the chance to just randomly generate a neural network which happens to be a fixpoint. However, among $100,000$ randomly generated nets for each type of reduction, we did not find a single fixpoint. Thus, we can clearly attribute the attraction towards **0** to self-application.

For the weightwise reduction $\lhd_{ww}$, it is rather easy to construct a non-zero fixpoint by hand: For a network $\mathcal{I}$, we set all leftmost weights per layer to 1 and all other weights to 0, thus implementing the identity function on the inputs of $\mathcal{I}$, which clearly fulfills the fixpoint property. This allows us to test if the non-zero fixpoints form an attractor in the weight space like the zero fixpoint does: We added small amounts of noise to all weights of $\mathcal{I}$ and then subjected the resulting network $\mathcal{J}$ to several steps of self-application, checking if $\mathcal{J}$ would remain stable around $\mathcal{I}$ or "verge", i.e., either converge towards **0** or diverge towards infinite weights. However, even adding just at most 10^{-9} noise to each weight eventually caused all networks to "verge". Figure 3 shows the experiment for various amounts of noise. Adding less noise unsurprisingly causes the network to longer fulfill the ε-fixpoint property and to "verge" later, which possibly means that the network fulfills the fixpoint property again when converging to **0** (but we did not count that).

Thus, while self-application on its own shows a stable intent to approach the fixpoint **0**, it does not seem capable of creating any other fixpoints.

Self-Training

Subjecting randomly generated neural networks to self-training with respect to the aggregating training reduction $\leftrightsquigarrow_{agg}$ yields results as shown in Figure 4. All networks evolve for a few steps of self-training, then their weights remain constant. Note that each network approaches a different point in the weight space. Most interestingly, these points are fixpoints, even though we only apply self-training and fixpoints are defined using self-application. Moreover, all of these fixpoints are non-zero.

Figure 5 shows a detailed analysis for all types of reduction: While recurrent neural networks still tend to diverge a lot, aggregating networks converge towards weights that do not represent a fixpoint. However, the weightwise networks converge to non-trivial fixpoints with utmost reliability.

In order to elaborate on the opportunities of interaction between self-application and self-training, we construct an experiment where the two appear in alternation. The results are shown in Figure 6: While aggregating networks reach the zero fixpoint so fast via self-application that self-training is not able to add anything to that, weightwise networks need about 200–300 steps of self-training between each self-application to converge to fixpoints as reliably.

Soups

As we have discussed several means of neural networks interacting with themselves, it seems a reasonable next step to open up these interactions and build a population of mutually interacting networks. A suitable combination of a population of individuals and various interactions is called *soup* and works like an artificial chemistry system (cf. Dittrich et al. (2001)). This means that a soup evolves over a fixed amount of epochs. At every epoch, several different interaction operators can be applied to networks in the population with a certain chance, resulting in new networks and thus a changed population.

Interaction 1 (Self-Train)**.** *Applied to every single network $\mathcal{N}$ for an amount of steps A, self-training substitutes its weights with $\mathcal{N}' = \mathcal{N} \underbrace{\leftrightsquigarrow \mathcal{N} \ldots \leftrightsquigarrow \mathcal{N}}_{A\ times}$.*

Interaction 2 (Attack)**.** *Applied to two random networks $\mathcal{M}, \mathcal{N}$ at a chance α, attacking substitutes the weights of the attacked network $\mathcal{M}$ with the weights given via $\mathcal{M}' = \mathcal{N} \lhd \mathcal{M}$.*

Intuitively, attacking applies the function represented by the network $\mathcal{N}$ to another network $\mathcal{M}$. Self-training remains basically unchanged from the non-soup scenario and provides a background evolution to every network in the population, even when it is not involved in any attack.

Figure 7 shows the evolution of a soup employing self-training and attacking. The networks start out randomly placed in the weight space and self-train towards fixpoints in the beginning. The big jumps in the networks' trajectories

Taken from original publication: Thomas Gabor, Steffen Illium, Andy Mattausch, Lenz Belzner, and Claudia Linnhoff-Popien. Self-replication in neural networks. In *Artificial Life Conference Proceedings*, pages 424–431. MIT Press, 2019

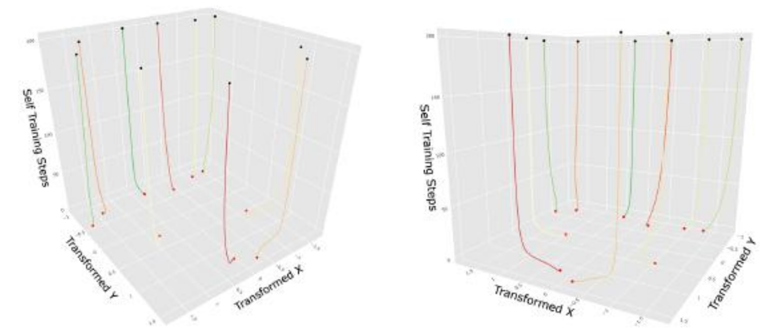


Figure 4: 10 independent runs of **self-training** with respect to the aggregating training reduction $\rightsquigarrow_{ww}$. 10 neural networks $\mathcal{N} : \mathbb{R}^4 \to \mathbb{R}$ with two hidden layers with two cells each were initialized randomly and then subjected to 200 steps of self-training each. The figure shows two perspectives on the same three-dimensional graph. The 20 weights in total per network were visualized in a two-dimensional space based on the transformed bases X and Y derived via PCA. All networks converge to different fixpoints with non-zero weights.

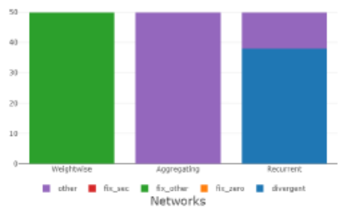


Figure 5: 50 independent runs of **self-training** each with respect to all three different types of reduction. We show an analysis of the networks (which were initialized randomly) after 1000 steps of self-training.

stem from being attacked by other networks; self-training then leads them to new fixpoints. Note that as self-training causes the networks to converge towards fixpoints, the impact of near-fixpoint networks' attacks becomes less and less. Most interestingly though, almost all attacks seem to drive the attacked networks towards the main cluster of the soup, where most networks gather in the end. This not only shows emergent behavior as the networks form a group as a cluster of fixpoints somehwere in the weight space (neither at the the center of mass from the initial population nor anywhere near **0**), but it can be also interpreted as a clear instance of (self-)replication within the networks of this soup.

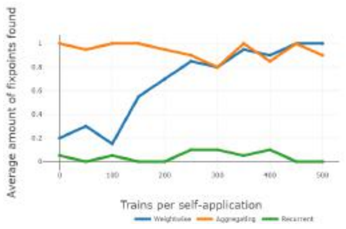


Figure 6: Evaluation of a mixed setting of self-application and self-training. For each type of reduction, 20 neural networks were generated at random and then subjected to 4 steps of self-application. In between those steps, $0, 50, ..., 500$ steps of self-training were executed (see x-axis). The y-axis shows the average ratio of fixpoints (both zero and non-zero) found out of all runs, where a value of 1 means that all runs resulted in a fixpoint.

In Figure 8, we further evaluate the impact of parameter A in Interaction 1 for both weightwise and aggregating neural networks. (As recurrent networks already did not show sufficient compatibility with application, we omit these results.) More self-training manages to stabilize the weightwise networks' ability to find non-zero fixpoints. Still, even in a soup setting, aggregating networks converge to **0** to a distinctive degree.

429

Taken from original publication: Thomas Gabor, Steffen Illium, Andy Mattausch, Lenz Belzner, and Claudia Linnhoff-Popien. Self-replication in neural networks. In *Artificial Life Conference Proceedings*, pages 424–431. MIT Press, 2019

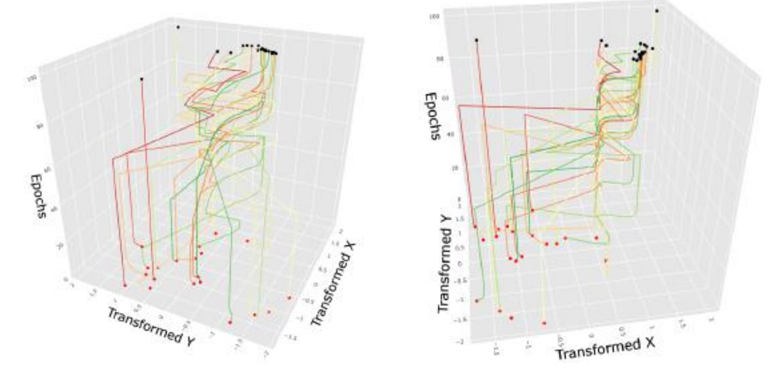


Figure 7: Run of one soup consisting of 20 neural networks using the weightwise reductions $\lhd_{ww}$ and $\leftrightsquigarrow_{ww}$. The 20 neural networks $\mathcal{N} : \mathbb{R}^4 \rightarrow \mathbb{R}$ with two hidden layers with two cells each were initialized randomly and then evolved for 100 epoch. Per epoch, every network had a chance of 0.1 to attack another network and was subjected to 30 steps of self-training. This setup allowed for emergent behavior of the network forming a cluster at a region fo all non-zero fixpoints. The figure shows two perspectives on the same three-dimensional graph. The 20 weights in total per network were visualized in a two-dimensional space based on the transformed bases X and Y derived via PCA.

Related Work

There is some research in generating neural networks using other neural networks (cf. Schmidhuber (1992); Stanley et al. (2009); Deutsch (2018), e.g.). However, without any suitable reduction operations, these approaches cannot be used to produce self-replicating structures.

Our results on self-application agree with Chang and Lipson (2018) on the weightwise reduction. We extended the experiments with several means of reduction and managed to find non-trivial, non-zero fixpoints up to a very low error ε by introducing our weightwise reduction in combination with our notion of self-training. We augmented the approach by studying the combination of self-application and self-training. However, the inclusion of auxiliary fitness functions has not been considered in our work.

The idea to generate fixpoints via repeated self-application is based on Fontana and Buss (1996), who showed the emergence of fixpoints from having random expressions in the λ-calculus interact. They, too, construct an artificial chemistry system based on their functional abstraction and see complex structures of fixpoints arise. Sadly, we did not observe higher-order fixpoints as they did for λ-expressions. Possible connections between λ-fixpoints or larger organizational structures in general and fixpoints in neural networks may still be explored (Larkin and Stocks (2004)).

In general, a vast amount of research is dedicated to artificial chemistry systems, utilizing very different representations for the particles in the soup: Dittrich et al. (2001) and Matsumaru et al. (2005) provide excellent overviews, to which we refer for the sake of brevity.

Conclusion

We have presented various reduction operations without any claim of completeness. Interesting reduction possibilities like extracting the main frequencies of the weight vector using a Fourier transformation are still to be tested to full extent. Most importantly, all settings, architectures and parameters of the neural networks we constructed still allow for more thorough exploration and evaluation in future work.

We have also performed some exploration of the distribution of fixpoints within the weight space by generating lots of non-trivial fixpoints using our setup of self-training. Especially discovering some kind of measurement of how rare fixpoints actually are and if they can occur in all regions of the weight space would be helpful.

We think that perhaps the most interesting contributions are the distinct behaviors observed in the soup made of neural networks (Figure 7). While we evaluated some parameters, there exist many different ways to evolve such a soup and many different interactions whose effects are yet to be explored. Early results on an interaction called *learn*, which

Taken from original publication: Thomas Gabor, Steffen Illium, Andy Mattausch, Lenz Belzner, and Claudia Linnhoff-Popien. Self-replication in neural networks. In *Artificial Life Conference Proceedings*, pages 424–431. MIT Press, 2019

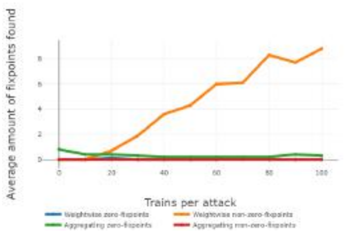


Figure 8: Evaluation of the impact of the number of trains per epoch on a soup consisting of 10 neural networks using the weightwise reduction $\lhd_{ww}$ or the aggregating reduction $\lhd_{agg}$. Averaged over 10 runs. Per epoch, every network had a chance of 0.1 to attack another network and was subjected to a fixed amount of steps of self-training (see the x-axis). The y-axis shows the amount of (zero or non-zero) fixpoints present in the final population of the soup.

substitutes the weights of the learning network $\mathcal{M}$ with the weights given via $\mathcal{M}' = \mathcal{M} \leftsquigarrow \mathcal{N}$ look most promising but were left out for brevity.

Eventually, we think that the dynamics of a soup might open up neural networks to a new kind of learning by not applying a goal function (and its respective loss) directly but by simply guiding a soup a certain way, perhaps achieving more diversity and robustness in the solutions reached (cf. Prokopenko (2013); Gabor et al. (2018), e.g.).

References

Bäck, T., Hammel, U., and Schwefel, H.-P. (1997). Evolutionary computation: Comments on the history and current state. *IEEE transactions on Evolutionary Computation*, 1(1):3–17.

Box, G. E. (1957). Evolutionary operation: A method for increasing industrial productivity. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 6(2):81–101.

Chang, O. and Lipson, H. (2018). Neural network quine. In *Artificial Life Conference Proceedings*. MIT Press.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Church, A. (1932). A set of postulates for the foundation of logic. *Annals of mathematics*, pages 346–366.

Conway, J. (1970). The game of life. *Scientific American*, 223(4):4.

Dawkins, R. (1976). *The Selfish Gene*. Oxford University Press, USA.

Deutsch, L. (2018). Generating neural networks with neural networks. *arXiv preprint arXiv:1801.01952*.

Dittrich, P. and Banzhaf, W. (1998). Self-evolution in a constructive binary string system. *Artificial Life*, 4(2):203–220.

Dittrich, P., Ziegler, J., and Banzhaf, W. (2001). Artificial chemistriesa review. *Artificial life*, 7(3):225–275.

Dorigo, M. and Di Caro, G. (1999). Ant colony optimization: a new meta-heuristic. In *Proceedings of the 1999 congress on evolutionary computation (CEC99)*, volume 2. IEEE.

Fontana, W. and Buss, L. (1996). The barrier of objects: from dynamical systems to bounded organizations.

Gabor, T., Belzner, L., Phan, T., and Schmid, K. (2018). Preparing for the unexpected: Diversity improves planning resilience in evolutionary algorithms. In *2018 IEEE International Conference on Autonomic Computing (ICAC)*. IEEE.

Hinton, G., Deng, L., Yu, D., Dahl, G., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Kingsbury, B., et al. (2012). Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal processing magazine*, 29.

Koza, J. R. (1994). Spontaneous emergence of self-replicating and evolutionarily self-improving computer programs. *Artificial life III*, 17:225–262.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*.

Larkin, J. and Stocks, P. (2004). Self-replicating expressions in the lambda calculus. In *Proceedings of the 27th Australasian conference on Computer science-Volume 26*, pages 167–173. Australian Computer Society, Inc.

Matsumaru, N., Centler, F., di Fenizio, P. S., and Dittrich, P. (2005). Chemical organization theory as a theoretical base for chemical computing. In *Proceedings of the 2005 Workshop on Unconventional Computing: From Cellular Automata to Wetware*, pages 75–88. Luniver Press.

Minsky, M. and Papert, S. (1972). *Perceptrons: An Introduction to Computational Geometry*. Mit Press.

Prokopenko, M. (2013). *Guided self-organization: Inception*, volume 9. Springer Science & Business Media.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

Schmidhuber, J. (1992). Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139.

Schoenholz, S. S., Pennington, J., and Sohl-Dickstein, J. (2017). A correspondence between random neural networks and statistical field theory. *arXiv preprint arXiv:1710.06570*.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017). Mastering the game of go without human knowledge. *Nature*, 550(7676):354.

Stanley, K. O., D'Ambrosio, D. B., and Gauci, J. (2009). A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15(2):185–212.

Turing, A. (1950). Computing machinery and intelligence. *Mind*, 59(236):433–460.

Taken from original publication: Thomas Gabor and Lenz Belzner. Genealogical distance as a diversity estimate in evolutionary algorithms. In *Measuring and Promoting Diversity in Evolutionary Algorithms (MPDEA@GECCO)*. ACM, 2017

Genealogical Distance as a Diversity Estimate in Evolutionary Algorithms

Thomas Gabor
LMU Munich
thomas.gabor@ifi.lmu.de

Lenz Belzner
LMU Munich
belzner@ifi.lmu.de

ABSTRACT

The evolutionary edit distance between two individuals in a population, i.e., the amount of applications of any genetic operator it would take the evolutionary process to generate one individual starting from the other, seems like a promising estimate for the diversity between said individuals. We introduce genealogical diversity, i.e., estimating two individuals' degree of relatedness by analyzing large, unused parts of their genome, as a computationally efficient method to approximate that measure for diversity.

CCS CONCEPTS

•Computing methodologies → Heuristic function construction; Genetic algorithms;

ACM Reference format:
Thomas Gabor and Lenz Belzner. 2017. Genealogical Distance as a Diversity Estimate
in Evolutionary Algorithms. In *Proceedings of GECCO '17 Companion, Berlin, Germany, July 15-19, 2017,* 6 pages.
DOI: http://dx.doi.org/10.1145/3067695.3082529

1 INTRODUCTION

Diversity has been a central point of research in the area of evolutionary algorithms. It is a well-known fact that maintaining a certain level of diversity aids the evolutionary process in preventing *premature convergence*, i.e., the phenomenon that the population focuses too quickly on a local optimum at hand and never reaches more fruitful areas of the fitness landscape [4, 14, 16]. Diversity thus plays a key role in adjusting the *exploration-exploitation trade-off* found in any kind of metaheuristic search algorithm.

We encountered this problem from an industry point of view when designing learning components for a system that needs to guarantee certain levels of quality despite being subjected to the probabilistic nature of its physical environment and probabilistic behavior of its machine learning parts [1]. Of course, any general solution for this kind of challenge is yet to be found. However, we believe that the engineering of (hopefully) reliable learning components can be supported by exposing all handles that search algorithms offer to the engineer at site. For scenarios like this one, we consider it most helpful to employ approaches that allow the engineer to actively control properties like diversity of the evolutionary search process instead of just observing diversity and adjust it indirectly via other parameters (like, e.g., the mutation rate).

Among the copious amount of different techniques to introduce diversity-awareness to evolutionary algorithms, many do not immediately make the job of adjusting a given evolutionary algorithm easier but instead require additional engineering effort: For example, one may need to define a distance metric specifically for the search domain at hand or adjust lots of hyperparameters in island or niching models. We thus attempt to define a more domain-independent and almost parameter-free measurement for diversity by utilizing the genetic operators already defined within any given evolutionary process.

We discuss related work in the following Section 2. We then explain the target metric called "evolutionary edit distance" in Section 3. Section 4 continues by introducing the notion of "genealogical diversity" as means to approximate that concept. We improve this approach in Section 5 by using a much simpler and computationally more efficient data structure. To support our ideas, we describe a basic evaluation scenario in which we have applied both approaches in Section 6. We end with a short conclusion in Section 7.

2 RELATED WORK

The importance of diversity for evolutionary algorithms is discussed throughout the body of literature on evolutionary computing ranging from entry level [4, 9] to specialized papers [13, 16]. In many cases, authors refer to diversity as a measure of the evolutionary algorithm's performance and try to configure the hyperparameters of the evolutionary algorithm as to achieve an optimal trade-off between exploration and exploration for the scenario at hand [15]. This measure can then be used to interact with the evolutionary process by adjusting its parameters [16] and/or actively altering the current population, for example via episodes of "hypermutation" [6] or migration of individuals from other (sub-)populations [11, 15]. On top of that, there exist a few approaches that include diversity into the evolutionary algorithm's objective function allowing us to use evolution's optimization abilities to explicitly achieve higher diversity of solutions [2].

An extensive overview of current approaches to increase diversity in evolutionary algorithms is provided in [14], which also defines a helpful taxonomy of said approaches. Whenever a diversity objective can be quantified, it can be used to build a classic multi-objective optimization problem and to apply the vast amount of techniques developed to solve these kinds of problems using evolutionary algorithms [7, 8, 10, 13].

Taken from original publication: Thomas Gabor and Lenz Belzner. Genealogical distance as a diversity estimate in evolutionary algorithms. In *Measuring and Promoting Diversity in Evolutionary Algorithms (MPDEA@GECCO)*. ACM, 2017

The authors of [17] address the very important issue of how to efficiently compute diversity estimates requiring to compare every individual of a population to every other. They develop an approach to reduce the complexity of said computation to linear time. However, it might still be interesting to analyze how well certain metrics scale even beyond that, as for example in the present paper we chose to sample the test set for diversity from the population to further save computation time.

3 EVOLUTIONARY EDIT DISTANCE

As described in the previous Section, there exists a vast amount of approaches to compute a population's diversity (and an individual's diversity with respect to that population). We found, among other things, that from an engineering point of view, many (if not most) of these approaches require the designer of the evolutionary algorithm to adjust certain functions or parameters based on the problem domain [5]. This gave rise to the idea of using the genetic operators already programmed for the problem domain to define a domain-independent notion of diversity.

The concept this line of thought is based on could be called *evolutionary edit distance*: Given two individuals x_1 and x_2 we want to estimate how many applications of a genetic operator it would take to turn one of these individuals into the other.[1] First, we start off by defining a lower bound on the number of operator applications, i.e., the *minimal evolutionary edit distance*.

We can assume that a given evolutionary process provides the genetic operator $o : \mathcal{D}^* \rightarrow \mathcal{D}$ where $\mathcal{D}$ is the problem domain in which our individuals live and $\mathcal{D}^*$ is a list of arbitrary many elements of $\mathcal{D}$. Most evolutionary algorithms define exactly two instances of genetic operators called mutation $m : \mathcal{D} \rightarrow \mathcal{D}$ and recombination $c : \mathcal{D} \times \mathcal{D} \rightarrow \mathcal{D}$, but we describe the more general case for now. However, in the general case genetic operators perform in a probabilistic manner, meaning that their exact results depend on chance. We describe this behavior mathematically by adding an index $s \in \mathcal{S}$ to o which represents the seed of a pseudo-random number generator (using seeds of type $\mathcal{S}$). Then, we can define the minimal evolutionary edit distance $mdist : \mathcal{D} \times \mathcal{D} \rightarrow \mathbb{N}$ as follows:

$$mdist(x_1, x_2) = \begin{cases} 0 & \textit{if } x_1 = x_2 \\ \min_{s \in \mathcal{S}} 1 + mdist(o_s(x_1), x_2) & \textit{otherwise} \end{cases}$$

Note that as long as we assume the genetic operator o to be symmetric (which they usually are), *mdist* is symmetric as well.

The minimal edit distance is not an accurate estimate of the actual effort it would take the evolutionary process to turn x_1 into x_2 since the required indexed instances o_s of the genetic operator o may be arbitrarily unlikely to occur in the process. Instead, we want to estimate the expected amount of applications of o given a realistic occurrence of instances of the genetic operator. Sadly, the complexity of this problem is equal to running an evolutionary algorithm optimizing its individuals to look like x_2 and thus potentially equally expensive regarding computational effort as the evolutionary process we are trying to augment with diversity.

However, if we want to use *mdist* to compute the diversity of individuals for a given evolutionary process, we never want to compare arbitrary solution candidates $x_1, x_2 \in \mathcal{D}$ but will only ever compare individuals within the current population $P \subseteq \mathcal{D}$ or at most individuals from the set X with $P \subseteq X \subseteq \mathcal{D}$, which is the set of all individuals ever generated by the evolutionary process. Each of those individuals has been generated through the repetitive application of the genetic operators already and so we have a set of concrete instances of o instead of having to reason about all o_s that *could* be used by the evolutionary process. We write the set of all actually generated instances of o as $\mathcal{O} = \{(x_0, o_{s_0}, x'_0), (x_1, o_{s_1}, x'_1) ..., (x_k, o_{s_k}, x'_k)\}$ where $k+1$ is the total amount of evolutionary operations performed and for all $(x_i, o_{s_i}, x'_i) \in \mathcal{O}$ the evolutionary process actually constructed $x'_i \in X$ by computing $o_{s_i}(x_i)$.

We can thus define the *factual* evolutionary edit distance $edist : X \times X \rightarrow \mathbb{N}$ as the total amount of operations it actually took to turn x into x':

$$edist(x_1, x_2) = \begin{cases} 0 & \textit{if } x_1 = x_2 \\ 1 & \textit{if } \exists s \in \mathcal{S} : (x_1, o_s, x_2) \in \mathcal{O} \\ 1 & \textit{if } \exists s \in \mathcal{S} : (x_2, o_s, x_1) \in \mathcal{O} \\ edist'(x_1, x_2) & \textit{otherwise} \end{cases}$$

$$\begin{aligned} edist'(x_1, x_2) = \min_{x \in X} \quad & 1 + \\ & \min_{s \in \mathcal{S}, (x, o_s, x_1) \in \mathcal{O}} \quad edist(o_s(x), x_1) + \\ & \min_{s \in \mathcal{S}, (x, o_s, x_2) \in \mathcal{O}} \quad edist(o_s(x), x_2)) \end{aligned}$$

Note that *edist* can only be defined this way when we assume that its parameters x_1 and x_2 have actually been generated through the application of genetic operators from a single base individual only. This is an unrealistic assumption: Completely unrelated individuals can be generated during evolution. Furthermore, defining the evolutionary edit distance this way requires multiple iterations through the whole set of X since we neglect many restrictions present in most genetic operators o.

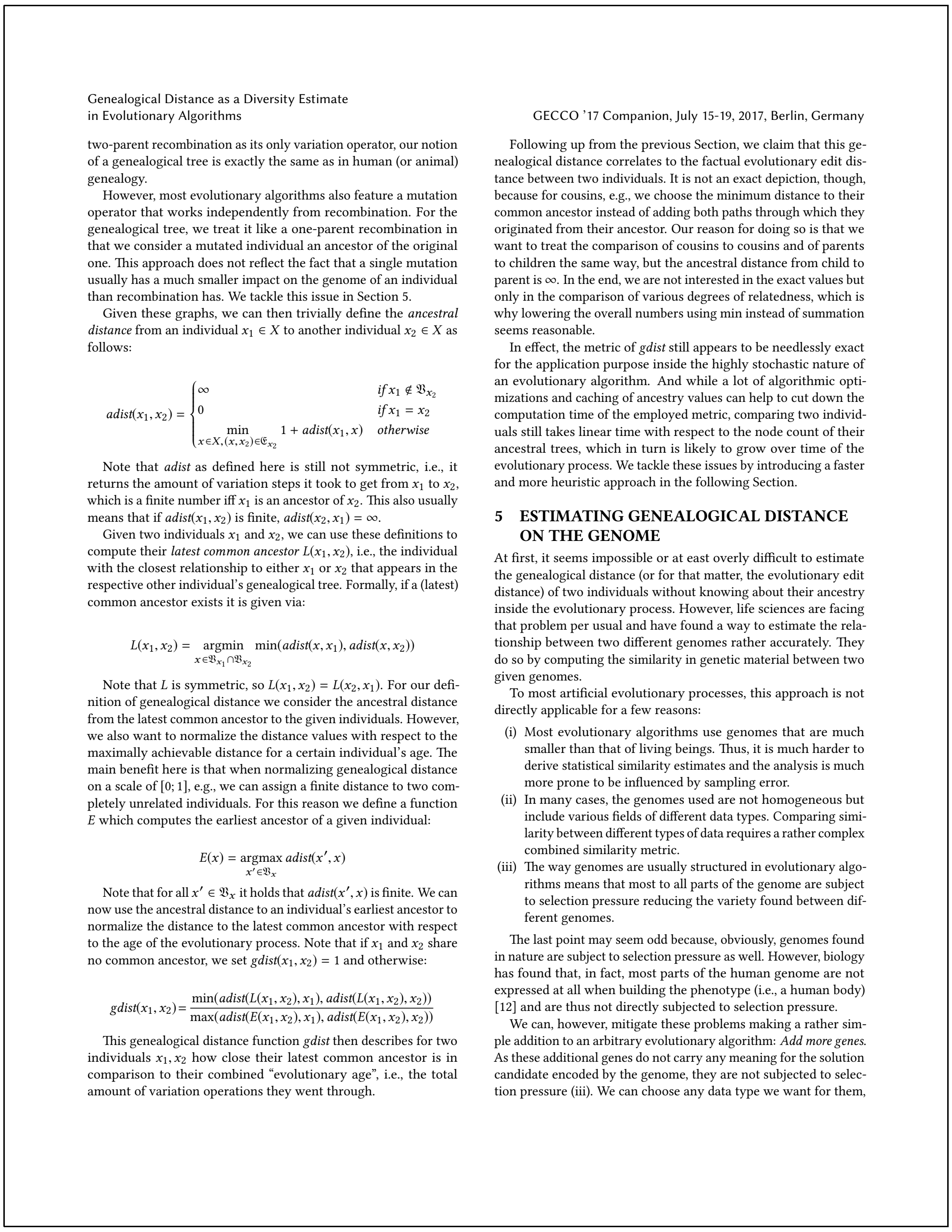
4 PATHS IN THE GENEALOGICAL TREE

In the context of evolutionary processes it seems natural to think of individuals as forming genealogical relationships between each other. These relations correspond to the genetic operators applied to an individual x to create the individual x'. Connecting all individuals (of all generations of the evolutionary process) to their respective children yields a directed, acyclic and usually non-connected graph. Starting from a single individual x, recursively traversing all incoming edges in reverse direction yields a connected subgraph containing all of x's ancestors. We call this graph the genealogical tree of x.

Formally, we write $\mathfrak{G}(x) = (\mathfrak{V}_x, \mathfrak{E}_x)$ for the genealogical tree of x consisting of vertices $\mathfrak{V}_x$ and edges $\mathfrak{E}_x$. For an evolutionary process producing (over all generations) the set of individuals X, it holds for all $x_1, x_2 \in X$ that $(x_1, x_2) \in \mathfrak{E}_{x_2}$ iff x_2 is the result of a variation of x_1. If we consider an evolutionary process with

[1]Because of the probabilistic nature of evolutionary algorithms, the evolutionary edit distance would actually be a distribution over integers. If a scalar value is needed, we could then compute the *expected evolutionary edit distance*.

Taken from original publication: Thomas Gabor and Lenz Belzner. Genealogical distance as a diversity estimate in evolutionary algorithms. In *Measuring and Promoting Diversity in Evolutionary Algorithms (MPDEA@GECCO)*. ACM, 2017



two-parent recombination as its only variation operator, our notion of a genealogical tree is exactly the same as in human (or animal) genealogy.

However, most evolutionary algorithms also feature a mutation operator that works independently from recombination. For the genealogical tree, we treat it like a one-parent recombination in that we consider a mutated individual an ancestor of the original one. This approach does not reflect the fact that a single mutation usually has a much smaller impact on the genome of an individual than recombination has. We tackle this issue in Section 5.

Given these graphs, we can then trivially define the *ancestral distance* from an individual $x_1 \in X$ to another individual $x_2 \in X$ as follows:

$$adist(x_1, x_2) = \begin{cases} \infty & if\, x_1 \notin \mathfrak{V}_{x_2} \\ 0 & if\, x_1 = x_2 \\ \min\limits_{x \in X, (x, x_2) \in \mathfrak{E}_{x_2}} 1 + adist(x_1, x) & otherwise \end{cases}$$

Note that *adist* as defined here is still not symmetric, i.e., it returns the amount of variation steps it took to get from x_1 to x_2, which is a finite number iff x_1 is an ancestor of x_2. This also usually means that if $adist(x_1, x_2)$ is finite, $adist(x_2, x_1) = \infty$.

Given two individuals x_1 and x_2, we can use these definitions to compute their *latest common ancestor* $L(x_1, x_2)$, i.e., the individual with the closest relationship to either x_1 or x_2 that appears in the respective other individual's genealogical tree. Formally, if a (latest) common ancestor exists it is given via:

$$L(x_1, x_2) = \operatorname*{argmin}_{x \in \mathfrak{V}_{x_1} \cap \mathfrak{V}_{x_2}} \min(adist(x, x_1), adist(x, x_2))$$

Note that L is symmetric, so $L(x_1, x_2) = L(x_2, x_1)$. For our definition of genealogical distance we consider the ancestral distance from the latest common ancestor to the given individuals. However, we also want to normalize the distance values with respect to the maximally achievable distance for a certain individual's age. The main benefit here is that when normalizing genealogical distance on a scale of $[0; 1]$, e.g., we can assign a finite distance to two completely unrelated individuals. For this reason we define a function E which computes the earliest ancestor of a given individual:

$$E(x) = \operatorname*{argmax}_{x' \in \mathfrak{V}_x} adist(x', x)$$

Note that for all $x' \in \mathfrak{V}_x$ it holds that $adist(x', x)$ is finite. We can now use the ancestral distance to an individual's earliest ancestor to normalize the distance to the latest common ancestor with respect to the age of the evolutionary process. Note that if x_1 and x_2 share no common ancestor, we set $gdist(x_1, x_2) = 1$ and otherwise:

$$gdist(x_1, x_2) = \frac{\min(adist(L(x_1, x_2), x_1), adist(L(x_1, x_2), x_2))}{\max(adist(E(x_1, x_2), x_1), adist(E(x_1, x_2), x_2))}$$

This genealogical distance function *gdist* then describes for two individuals x_1, x_2 how close their latest common ancestor is in comparison to their combined "evolutionary age", i.e., the total amount of variation operations they went through.

Following up from the previous Section, we claim that this genealogical distance correlates to the factual evolutionary edit distance between two individuals. It is not an exact depiction, though, because for cousins, e.g., we choose the minimum distance to their common ancestor instead of adding both paths through which they originated from their ancestor. Our reason for doing so is that we want to treat the comparison of cousins to cousins and of parents to children the same way, but the ancestral distance from child to parent is ∞. In the end, we are not interested in the exact values but only in the comparison of various degrees of relatedness, which is why lowering the overall numbers using min instead of summation seems reasonable.

In effect, the metric of *gdist* still appears to be needlessly exact for the application purpose inside the highly stochastic nature of an evolutionary algorithm. And while a lot of algorithmic optimizations and caching of ancestry values can help to cut down the computation time of the employed metric, comparing two individuals still takes linear time with respect to the node count of their ancestral trees, which in turn is likely to grow over time of the evolutionary process. We tackle these issues by introducing a faster and more heuristic approach in the following Section.

5 ESTIMATING GENEALOGICAL DISTANCE ON THE GENOME

At first, it seems impossible or at east overly difficult to estimate the genealogical distance (or for that matter, the evolutionary edit distance) of two individuals without knowing about their ancestry inside the evolutionary process. However, life sciences are facing that problem per usual and have found a way to estimate the relationship between two different genomes rather accurately. They do so by computing the similarity in genetic material between two given genomes.

To most artificial evolutionary processes, this approach is not directly applicable for a few reasons:

(i) Most evolutionary algorithms use genomes that are much smaller than that of living beings. Thus, it is much harder to derive statistical similarity estimates and the analysis is much more prone to be influenced by sampling error.
(ii) In many cases, the genomes used are not homogeneous but include various fields of different data types. Comparing similarity between different types of data requires a rather complex combined similarity metric.
(iii) The way genomes are usually structured in evolutionary algorithms means that most to all parts of the genome are subject to selection pressure reducing the variety found between different genomes.

The last point may seem odd because, obviously, genomes found in nature are subject to selection pressure as well. However, biology has found that, in fact, most parts of the human genome are not expressed at all when building the phenotype (i.e., a human body) [12] and are thus not directly subjected to selection pressure.

We can, however, mitigate these problems making a rather simple addition to an arbitrary evolutionary algorithm: *Add more genes*. As these additional genes do not carry any meaning for the solution candidate encoded by the genome, they are not subjected to selection pressure (iii). We can choose any data type we want for them,

Taken from original publication: Thomas Gabor and Lenz Belzner. Genealogical distance as a diversity estimate in evolutionary algorithms. In *Measuring and Promoting Diversity in Evolutionary Algorithms (MPDEA@GECCO)*. ACM, 2017

so we can adhere to a type that allows for an easy comparison between individuals (ii). And finally, we can choose a comparatively large size for these genes so that they allow for a subtle comparison (i). For the lack of a better name, we call these additional genes by the name *trash genes* to emphasize that they do not directly contribute to the individual's fitness.

For our experiments thus far, we have chosen a simple bit vector of a fixed length τ to encode the added trash genes. Choosing τ too small ($2^\tau < n$ where n is the population size) can obviously be detrimental to the distance estimate, but choosing very large τ ($2^\tau \gg n$) has not shown any negative effects in our preliminary experiments. Using bit vectors comes with the advantage that genetic operators like mutation and recombination are trivially defined on this kind of data structure.

Formally, to any individual $x \in X$ we assign a bit vector $T(x) = \langle t_0, ..., t_{\tau-1} \rangle$ with $t_i \in \{0, 1\}$ for all i, which is initialized at random when the individual x is created. Every time a mutation operation is performed on x, we perform a random single bit flip on $T(x)$.[2] For each recombination of x_1 and x_2, we generate the child's trash bit vector via uniform crossover of $T(x_1)$ and $T(x_2)$.

We can then compute a trash bit distance *tdist* between two individuals x_1 and x_2 simply by returning the Hamming distance between their respective trash genes:

$$\begin{aligned} tdist(x_1, x_2) &= \frac{1}{\tau} * H(T(x_1), T(x_2)) \\ &= \frac{1}{\tau} * \sum_{i=0}^{\tau-1} |T(x_1)_i - T(x_2)_i| \end{aligned}$$

This metric clearly is symmetric. Again, we normalize the output by dividing it by τ. Furthermore, trash bit vectors allow for a more detailed distinction between the impact of various genetic operators: The expected distance between two randomly generated individuals x_1 and x_2 is $\mathbb{E}(tdist(x_1, x_2)) = 0.5$. However, the distance between parents and children is reasonably lower: If x is the result of mutating x', we expect their trash bit distance to be $\mathbb{E}(tdist(x, x')) = 1/\tau$. The trash bit distance between a parent x' of a recombination operator and its child x is $\mathbb{E}(tdist(x, x')) = 0.25$ since the child shares about half of its trash bits with this one parent x' by the nature of crossover, resulting in a Hamming distance of 0 on this subset, and the other half with the other parent, say x'', with which the first parent x' naturally shares about half of its trash bits when no other assumptions about the parents' ancestry apply. This means that for the subset of trash bits inherited from x'', x and x' have a trash bit distance of 0.5, resulting in a 0.25 average for the whole bit vector of x.[3] If the parents are related (or share improbable amounts of trash bits by chance), lower numbers for *tdist* can be achieved.

These examples should illustrate that the computed trash bit diversity is able to express genealogical relations between individuals. It stresses recombination over mutation but in doing so reflects the impact the respective operators have on the individual's actual genome. We thus propose trash bit vectors as a much simpler and more efficient implementation of genealogical diversity.

As is clear from the usage of the "expected value" $\mathbb{E}$ in these computations, the actual distance between parents and offspring is now always subject to random effects. However, so is their similarity on the non-trash genes as well.[4] This kind of probabilistic behavior is an intrinsic part of evolutionary algorithms. However, it may make sense to base the recombination on the trash bit vector on the recombination of the non-trash genes so that probabilistic tendencies are kept in sync. This is up to future research.

Finally, the computational effort to compute the trash bit distance is at most times negligible. Computing the distance between two individuals is an operation that can be performed in $O(\tau)$ and while we expect there to be a lose connection between population size and the optimal τ, for a given evolution process with a fixed population size, this means that trash bit diversity can be computed in constant time. Trivially, this also means the concept scales with population size and age.

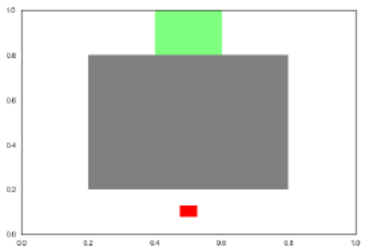


Figure 1: Illustration of the setup of the scenario. Marked in red is the starting position of the agent. The green area defines the goal which the agent is supposed to drive to. The gray area is the main obstacle the robot needs to drive around.

6 EXPERIMENT

To verify the practical applicability of the concept of genealogical diversity and its realizations presented in the previous Sections 4 and 5, respectively, we constructed a simple experimental setup: We define a simple routing task in which a robot has to choose a sequence of 10 continuous actions $a \in \mathbb{R} \times \mathbb{R}$ to reach a marked area. Each action takes exactly one time step and can move the robot across a Manhattan square of 0.5 at most. For each time step the robot remains inside the designated target area, it is rewarded a bonus of +1. In order to reach that area, the robot has to find a path around an obstacle blocking the direct way. Figure 1 shows a simple visualization of the setup described here.

[2]Note that this works for typical mutation operators on the non-trash genes, which tend to change very little about the genome as well. If more invasive mutation operators are employed, a likewise operation on the bit vector could be provided.

[3]These numbers correspond closely to the degrees of genetic relationship mentioned in [3].

[4]For example, a child generated via uniform crossover has very slim chance of not inheriting any gene material from one parent at all. The same effect can now happen not only on the fitness-relevant genes but also on the genes used for diversity marking.

Taken from original publication: Thomas Gabor and Lenz Belzner. Genealogical distance as a diversity estimate in evolutionary algorithms. In *Measuring and Promoting Diversity in Evolutionary Algorithms (MPDEA@GECCO)*. ACM, 2017

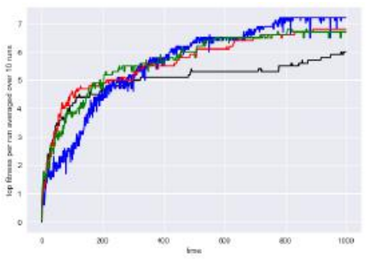


Figure 2: Average fitness achieved over time by the the non-diverse genetic algorithm (black), the domain-specific diverse genetic algorithm (blue), the genealogically diverse algorithm based on the genealogical tree distance (green) and the genealogically diverse genetic algorithm using the trash bit distance (red). To mitigate random effects a bit, the fitness values have been averaged over 10 complete evolution runs.

We solved this scenario with four different evolutionary algorithms. All of these use a population size of 20 individuals and have been executed for 1000 generations. For this kind of continuous optimization problem, that is not enough time for them to fully converge. We constructed a standard setup of an evolutionary algorithm with a continuous mutation operator working on a single action at a time and activated with a probability of 0.2. We employ uniform crossover with a probability of 0.3 per individual. A recombination partner is selected from a two-player tournament and offspring is added to the population before the selection step. Furthermore, 2 new individuals per generation are generated randomly.

Within this setup, we define a standard genetic algorithm using a fitness function that simply returns the aforementioned bonus for each individual. It performs well but seems to suffer from premature convergence in this setup (see Figure 2 for all plots). This is the baseline approach all diversity-enabled versions of the genetic algorithm can be tested against.

To introduce the diversity of the solutions to the genetic algorithm, we choose the approach to explicitly include the distance of the individual x to other individuals of P in x's fitness. But we do not construct a multi-objective optimization problem (as in [10, 13], e.g.) but simply define a weighting function to flatten these objectives. Formally, given the original fitness function f and an average diversity measure d of a single individual with respect to the population $P \subseteq X$, we define an adapted fitness function f' as follows:

$$f'(x, P) = f(x) + \lambda * d(x, P)$$

It is important to note that while we use f' for the purpose of selection inside the evolutionary algorithm, all external analysis (plotting, e.g.) is performed on the value of f only in order to keep the results comparable. Also note that we reduce the computational effort to calculate any distance metric d used in this paper by not evaluating a given individual's diversity against the whole population P but only against a randomly chosen subset of 5 individuals. In our experiments, this approach has been sufficiently stable.

Furthermore, we determined the optimal λ for each algorithm using grid search on this hyperparameter. In a scenario like this, where higher diversity yields better results overall, it appears reasonable to think that λ could be determined adaptively during the evolutionary process. This is still up to further research.

For evaluation purposes, we provided a domain-specific distance function. In this simple scenario, this can be defined quickly as well and we chose to use the sum of all differences between actions at the same position in the sequence. Figure 2 shows that this approach takes a bit longer to learn but can then evade local optima better, showing a curve that we would expect from a more diverse genetic algorithm.

Finally, we implemented both genealogical distance metrics presented in this paper. We can see in Figure 2 that both approaches in fact perform comparably, even though trash bit vectors require much less computational effort. For this experiment, we used $\tau = 32$.

7 CONCLUSION

In this paper, we have introduced the expected evolutionary edit distance as a promising target for diversity-aware optimization within evolutionary algorithms. Having found that it cannot be reasonably computed within another evolutionary process, we developed approaches to estimate that distance more efficiently. To do so, we introduced the notion of genealogical diversity and presented a method to estimate it accurately using very little computational resources.

The experimental results show the initial viability of the approach used here and allow for many future applications. Some of these have been realized in [5]. Other promising directions for future work have been mentioned throughout and include plans to omit the hyperparameter λ by using genealogical diversity within a true multi-objective evolutionary algorithm or by opening λ for self-adaptation by the evolutionary process. Furthermore, from a biological point of view, a genealogical selection process is less common in survivor selection than it is in parent selection. Testing if the metaphor to biology holds in that case would be an immediate next step of research.

REFERENCES

[1] Lenz Belzner, Michael Till Beck, Thomas Gabor, Harald Roelle, and Horst Sauer. 2016. Software engineering for distributed autonomous real-time systems. In *Proceedings of the 2nd International Workshop on Software Engineering for Smart Cyber-Physical Systems*. ACM, 54–57.

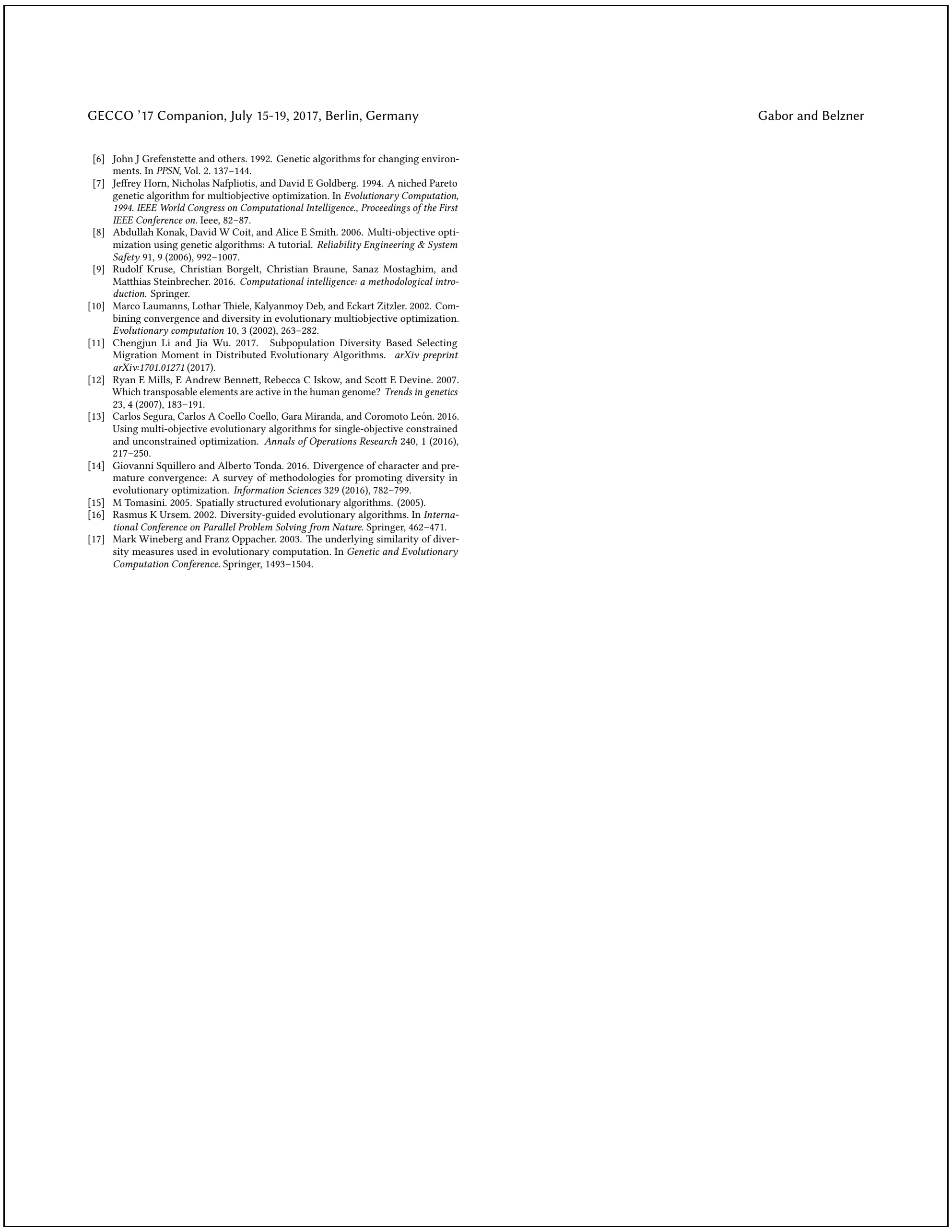
[2] Markus Brameier and Wolfgang Banzhaf. 2002. Explicit control of diversity and effective variation distance in linear genetic programming. In *European Conference on Genetic Programming*. Springer, 37–49.

[3] Richard Dawkins and others. 2016. *The selfish gene*. Oxford university press.

[4] Agoston E Eiben and James E Smith. 2003. *Introduction to evolutionary computing*. Vol. 53. Springer.

[5] Thomas Gabor. 2017. Preparing for the Unexpected: Diversity Improves Resilience in Online Genetic Algorithms. In *Proceedings of The 14th IEEE International Conference on Autonomic Computing (ICAC2017)*. Submitted.

Taken from original publication: Thomas Gabor and Lenz Belzner. Genealogical distance as a diversity estimate in evolutionary algorithms. In *Measuring and Promoting Diversity in Evolutionary Algorithms (MPDEA@GECCO)*. ACM, 2017



[6] John J Grefenstette and others. 1992. Genetic algorithms for changing environments. In *PPSN*, Vol. 2. 137–144.

[7] Jeffrey Horn, Nicholas Nafpliotis, and David E Goldberg. 1994. A niched Pareto genetic algorithm for multiobjective optimization. In *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*. Ieee, 82–87.

[8] Abdullah Konak, David W Coit, and Alice E Smith. 2006. Multi-objective optimization using genetic algorithms: A tutorial. *Reliability Engineering & System Safety* 91, 9 (2006), 992–1007.

[9] Rudolf Kruse, Christian Borgelt, Christian Braune, Sanaz Mostaghim, and Matthias Steinbrecher. 2016. *Computational intelligence: a methodological introduction*. Springer.

[10] Marco Laumanns, Lothar Thiele, Kalyanmoy Deb, and Eckart Zitzler. 2002. Combining convergence and diversity in evolutionary multiobjective optimization. *Evolutionary computation* 10, 3 (2002), 263–282.

[11] Chengjun Li and Jia Wu. 2017. Subpopulation Diversity Based Selecting Migration Moment in Distributed Evolutionary Algorithms. *arXiv preprint arXiv:1701.01271* (2017).

[12] Ryan E Mills, E Andrew Bennett, Rebecca C Iskow, and Scott E Devine. 2007. Which transposable elements are active in the human genome? *Trends in genetics* 23, 4 (2007), 183–191.

[13] Carlos Segura, Carlos A Coello Coello, Gara Miranda, and Coromoto León. 2016. Using multi-objective evolutionary algorithms for single-objective constrained and unconstrained optimization. *Annals of Operations Research* 240, 1 (2016), 217–250.

[14] Giovanni Squillero and Alberto Tonda. 2016. Divergence of character and premature convergence: A survey of methodologies for promoting diversity in evolutionary optimization. *Information Sciences* 329 (2016), 782–799.

[15] M Tomasini. 2005. Spatially structured evolutionary algorithms. (2005).

[16] Rasmus K Ursem. 2002. Diversity-guided evolutionary algorithms. In *International Conference on Parallel Problem Solving from Nature*. Springer, 462–471.

[17] Mark Wineberg and Franz Oppacher. 2003. The underlying similarity of diversity measures used in evolutionary computation. In *Genetic and Evolutionary Computation Conference*. Springer, 1493–1504.

Taken from original publication: Thomas Gabor, Lenz Belzner, and Claudia Linnhoff-Popien. Inheritance-based diversity measures for explicit convergence control in evolutionary algorithms. In *The Genetic and Evolutionary Computation Conference (GECCO)*, 2018

Inheritance-Based Diversity Measures for Explicit Convergence Control in Evolutionary Algorithms

Thomas Gabor, Lenz Belzner, Claudia Linnhoff-Popien
LMU Munich

ABSTRACT

Diversity is an important factor in evolutionary algorithms to prevent premature convergence towards a single local optimum. In order to maintain diversity throughout the process of evolution, various means exist in literature. We analyze approaches to diversity that (a) have an explicit and quantifiable influence on fitness at the individual level and (b) require no (or very little) additional domain knowledge such as domain-specific distance functions. We also introduce the concept of genealogical diversity in a broader study. We show that employing these approaches can help evolutionary algorithms for global optimization in many cases.

CCS CONCEPTS

• **Computing methodologies → Heuristic function construction**; **Randomized search**; • **Theory of computation** → *Random search heuristics*; Sample complexity and generalization bounds;

KEYWORDS

diversity, evolutionary algorithms, premature convergence, optimization, genetic drift

ACM Reference Format:
Thomas Gabor, Lenz Belzner, Claudia Linnhoff-Popien. 2018. Inheritance-Based Diversity Measures for Explicit Convergence Control in Evolutionary Algorithms. In *GECCO '18: Genetic and Evolutionary Computation Conference, July 15–19, 2018, Kyoto, Japan.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3205455.3205630

1 INTRODUCTION

Diversity has traditionally been known as key asset for an evolutionary process. Higher levels of diversity within the population undergoing evolution steer the focus of the evolutionary search towards the exploration of the search space and away from convergence on the already found solutions. This often helps discover better global solutions. The precise handling of the exploration/exploitation dilemma is of central importance for the the success of all importance sampling techniques [4, 20].

Thus, many methods have been suggested to observe and subsequently control the level of diversity within a population. These have also been compared in previous studies [21]. All them introduce a notion of diversity given either (a) as a function of the population or (b) as a relation between an individual and (parts of) the population. This means that we can use diversity measures of type (a) to motivate and evaluate global approaches to increase or decrease the levels of diversity (like an increase of hyper-mutation or migration). Measures of type (b) can estimate the diversity introduced by each single individual and are commonly used as an additional objective to the evolutionary process. Instead of using a classical multi-objective evolutionary algorithm each time we employ individual-based diversity, we can also adjust the selection process to respect diversity in a different manner or use other common techniques to transform multi-objective evolutionary algorithms into a single-objective case.

In this paper, we focus on a specific kind of individual-based diversity: In order to avoid defining (and assessing) domain-specific measures of diversity that need to be adjusted to the specific data types used for the individuals' genomes, we attempt to create a more general approach to diversity that does not directly depend on the structure or contents of the genomes. Instead, we want to leverage information already generated by the evolutionary process in order to give an estimate of the diversity of specific individuals. To this end, we perform a thorough evaluation of the novel notion of *genealogical diversity* [10]: We track the genealogical relations between individuals (in an efficient manner) and then assume that closely related individuals are more similar than unrelated individuals regarding the diversity they add towards the population. We show that evolutionary algorithms using genealogical diversity can reach similar levels of performance as those using domain-specific diversity measures and usually achieve better results than other generic approaches to diversity (such as population ensembles).

Our research is originally motivated from a software engineering point of view: Self-adaptation and self-organization are playing an increasingly important role in the design and implementation of large-scale software and cyber-physical systems, the reason being that state-of-the-art methods of optimization are not only able to save effort for human developers but are starting to show the ability to forge solutions that allow for entirely new applications [5, 25]. However, many of the new techniques for autonomous search come with a large amount of parameters that are expensive to fully evaluate. We thus see an inherent benefit from providing means to control the convergence of an evolutionary search process without depending on a specific choice of data structure or search domain.

A short overview of related work is given in the following Section 2. We introduce some diversity techniques in greater detail in Section 3. We thereby motivate and introduce the approach of genealogical diversity. Section 4 discusses empirical experiments that justify our approach. Finally, Section 5 provides a recap on this paper and a glimpse onto future work.

Taken from original publication: Thomas Gabor, Lenz Belzner, and Claudia Linnhoff-Popien. Inheritance-based diversity measures for explicit convergence control in evolutionary algorithms. In *The Genetic and Evolutionary Computation Conference (GECCO)*, 2018

2 RELATED WORK

Diversity is a topic often researched and discussed in literature about evolutionary algorithms. In short, a diverse population features many and by tendency more different genotypes of individuals, which has been known to be a key factor in preventing the problem of *premature convergence* [8]. However, oftentimes diversity is only regarded as a tool for an *a posteriori* analysis of the behavior of an evolutionary process [20]. Thus, a lot of research has been focused on constructing the evolutionary algorithm in such a way that low diversity is prevented implicitly *by design*. Approaches like island-based models or other spatial structures imposed on the set of individuals can be regarded as designs aiding higher diversity [22]. Ensemble methods open up a variety of configurations that can be used by the designer of the evolutionary algorithm to provably increase performance [13]. We deliberately choose a quite simple instance for a comparison in this paper. The full scope of combining the possibilities of diversity-awareness with ensemble learning are up to further research.

The maintenance of population diversity can also be tackled more explicitly: Observing diversity while the evolutionary process is still running allows a watchdog process to intervene whenever it does not fulfill the desired level of diversity [7, 23]. In such a setup, diversity is only improved by drastic methods altering the whole course of the evolutionary process in the form of a "last resort."

A newer line of research has focused on utilizing the evolutionary process itself to optimize diversity throughout the whole process, i.e., add diversity as a direct objective for the evolutionary algorithm [18]. This exposes the meta-goal of preventing premature convergence to the evolutionary algorithm and allows engineers to explicitly slow down the convergence process. Naturally, applying a second objective function yields a *multi-objective evolutionary algorithm (MOEA)*, which requires more complex (and thus more computationally expensive) methods of selection [15].

A most extensive overview of techniques of adjusting the exploration/exploitation trade-off in evolutionary algorithms can be found in [4]. The authors of [18] also provide a great overview over various methods to estimate the diversity of a single individual, remarking that distance-based methods have been shown to work best. The described distance functions like the distance of the closest neighbor, however, require consideration of most if not all of the individuals in the population, which comes with substantial computation load for large-scale examples. The authors of [21] perform an extensive survey of various means to define, measure and augment diversity in evolutionary algorithms. All of these papers also introduce comprehensive taxonomies.

For Particle Swarm Optimization and Differential Evolution approaches the authors of [19] have traced and visualized historic or genealogical relationships of individuals. However, they do not use that knowledge to further steer the evolutionary process.

3 DIVERSITY IN EVOLUTIONARY ALGORITHMS

As genetic algorithms maintain a pool of solution candidates at any given point in time, they are a natural fit for an optimization algorithm that searches for multiple local optima at once. However, even whole populations of solution candidates tend to converge to one single local optimum in some scenarios. A remedy discovered in the context of genetic algorithms is the notion of *diversity*: As we shortly discussed in the Introduction, diversity-enhancing techniques exist at the population level or at the individual level.

In this Section, we introduce our measure of genealogical diversity by deriving it from other measures of diversity we sketch in the course of this Section. First of all, however, we give a short definition of the formal framework we use for evolutionary algorithms.

3.1 Evolutionary Algorithms

Let D be the search domain of a given problem we want to optimize. The optimality of a solution candidate $x \in D$ is given via the *objective goal function* $g : D \to \mathbb{R}$. The solution to a maximization problem can thus be written as $\operatorname{argmax}_{x \in D} g(x)$ and likewise for minimization. The objective goal function will usually be the main influence on the fitness of an individual. In general, we define a fitness function $f : D \times \mathcal{P}(D) \to \mathbb{R}$ so that $f(x, P)$ denotes the fitness of the individual x within the population P with $x \in P$. Note that we pass on the whole population to the fitness function so that we can, e.g., respect the diversity of the individual with regard to said population. This also has the immediate effect that the fitness of an individual may vary without any actual change to the genome.

A population is a set of individuals. An individual usually directly represents a solution candidate $x \in D$ so we will use these notions interchangeably. We can thus give the type of a population P as $P \in \mathcal{P}(D)$. In detail, however, an individual is always part of a population and may thus have additional properties like genealogical relations such as, e.g., parents and children. A population is affected by evolutionary operators $o : \mathcal{P}(D) \to \mathcal{P}(D)$. In the evolutionary algorithms described in this paper we use common implementations of recombination, mutation, hypermutation and selection in that order. A series of populations resulting from the iterated application of these evolutionary operators is called an evolutionary process.

The examples in this paper show different instances of an evolutionary algorithm: The most simple one is called *non-diverse* and is directly derived from the setup described so far. Its fitness function f can simply be defined as

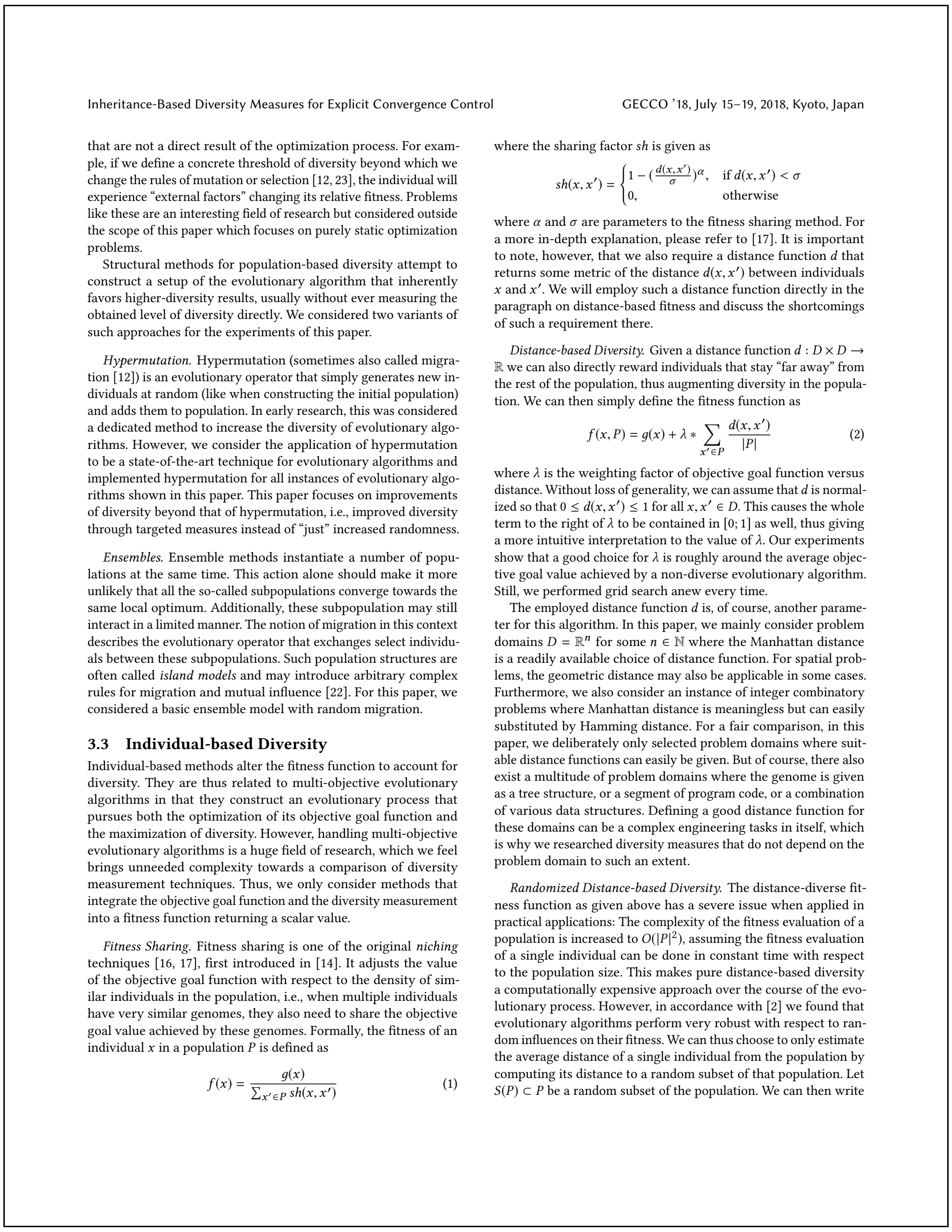
$$f(x, P) = g(x).$$

Note that when not stated otherwise, for the remainder of this Section we assume to optimize a maximization problem, i.e., maximize the fitness function. All definitions can be trivially adapted to the minimization case.

3.2 Population-based Diversity

Population-based methods attempt to increase the diversity within the evolutionary search process without computing a specific diversity value for every single individual. We further discern them into *structural* and *reactive* methods. The latter usually observe some diversity measurement throughout the evolutionary process and employ some methods to increase diversity once a state of little diversity has been observed. Commonly, these measures could be to increase the rate of mutation or hypermutation. Thus, reactive methods inevitably give rise to a dynamic optimization problem, i.e., they model changes to the setup of the evolutionary algorithm

Taken from original publication: Thomas Gabor, Lenz Belzner, and Claudia Linnhoff-Popien. Inheritance-based diversity measures for explicit convergence control in evolutionary algorithms. In *The Genetic and Evolutionary Computation Conference (GECCO)*, 2018



that are not a direct result of the optimization process. For example, if we define a concrete threshold of diversity beyond which we change the rules of mutation or selection [12, 23], the individual will experience "external factors" changing its relative fitness. Problems like these are an interesting field of research but considered outside the scope of this paper which focuses on purely static optimization problems.

Structural methods for population-based diversity attempt to construct a setup of the evolutionary algorithm that inherently favors higher-diversity results, usually without ever measuring the obtained level of diversity directly. We considered two variants of such approaches for the experiments of this paper.

Hypermutation. Hypermutation (sometimes also called migration [12]) is an evolutionary operator that simply generates new individuals at random (like when constructing the initial population) and adds them to population. In early research, this was considered a dedicated method to increase the diversity of evolutionary algorithms. However, we consider the application of hypermutation to be a state-of-the-art technique for evolutionary algorithms and implemented hypermutation for all instances of evolutionary algorithms shown in this paper. This paper focuses on improvements of diversity beyond that of hypermutation, i.e., improved diversity through targeted measures instead of "just" increased randomness.

Ensembles. Ensemble methods instantiate a number of populations at the same time. This action alone should make it more unlikely that all the so-called subpopulations converge towards the same local optimum. Additionally, these subpopulation may still interact in a limited manner. The notion of migration in this context describes the evolutionary operator that exchanges select individuals between these subpopulations. Such population structures are often called *island models* and may introduce arbitrary complex rules for migration and mutual influence [22]. For this paper, we considered a basic ensemble model with random migration.

3.3 Individual-based Diversity

Individual-based methods alter the fitness function to account for diversity. They are thus related to multi-objective evolutionary algorithms in that they construct an evolutionary process that pursues both the optimization of its objective goal function and the maximization of diversity. However, handling multi-objective evolutionary algorithms is a huge field of research, which we feel brings unneeded complexity towards a comparison of diversity measurement techniques. Thus, we only consider methods that integrate the objective goal function and the diversity measurement into a fitness function returning a scalar value.

Fitness Sharing. Fitness sharing is one of the original *niching* techniques [16, 17], first introduced in [14]. It adjusts the value of the objective goal function with respect to the density of similar individuals in the population, i.e., when multiple individuals have very similar genomes, they also need to share the objective goal value achieved by these genomes. Formally, the fitness of an individual x in a population P is defined as

$$f(x) = \frac{g(x)}{\sum_{x' \in P} sh(x, x')} \tag{1}$$

where the sharing factor sh is given as

$$sh(x, x') = \begin{cases} 1 - (\frac{d(x,x')}{\sigma})^{\alpha}, & \text{if } d(x,x') < \sigma \\ 0, & \text{otherwise} \end{cases}$$

where α and σ are parameters to the fitness sharing method. For a more in-depth explanation, please refer to [17]. It is important to note, however, that we also require a distance function d that returns some metric of the distance $d(x, x')$ between individuals x and x'. We will employ such a distance function directly in the paragraph on distance-based fitness and discuss the shortcomings of such a requirement there.

Distance-based Diversity. Given a distance function $d : D \times D \longrightarrow \mathbb{R}$ we can also directly reward individuals that stay "far away" from the rest of the population, thus augmenting diversity in the population. We can then simply define the fitness function as

$$f(x, P) = g(x) + \lambda * \sum_{x' \in P} \frac{d(x, x')}{|P|} \tag{2}$$

where λ is the weighting factor of objective goal function versus distance. Without loss of generality, we can assume that d is normalized so that $0 \leq d(x, x') \leq 1$ for all $x, x' \in D$. This causes the whole term to the right of λ to be contained in $[0; 1]$ as well, thus giving a more intuitive interpretation to the value of λ. Our experiments show that a good choice for λ is roughly around the average objective goal value achieved by a non-diverse evolutionary algorithm. Still, we performed grid search anew every time.

The employed distance function d is, of course, another parameter for this algorithm. In this paper, we mainly consider problem domains $D = \mathbb{R}^n$ for some $n \in \mathbb{N}$ where the Manhattan distance is a readily available choice of distance function. For spatial problems, the geometric distance may also be applicable in some cases. Furthermore, we also consider an instance of integer combinatory problems where Manhattan distance is meaningless but can easily substituted by Hamming distance. For a fair comparison, in this paper, we deliberately only selected problem domains where suitable distance functions can easily be given. But of course, there also exist a multitude of problem domains where the genome is given as a tree structure, or a segment of program code, or a combination of various data structures. Defining a good distance function for these domains can be a complex engineering tasks in itself, which is why we researched diversity measures that do not depend on the problem domain to such an extent.

Randomized Distance-based Diversity. The distance-diverse fitness function as given above has a severe issue when applied in practical applications: The complexity of the fitness evaluation of a population is increased to $O(|P|^2)$, assuming the fitness evaluation of a single individual can be done in constant time with respect to the population size. This makes pure distance-based diversity a computationally expensive approach over the course of the evolutionary process. However, in accordance with [2] we found that evolutionary algorithms perform very robust with respect to random influences on their fitness. We can thus choose to only estimate the average distance of a single individual from the population by computing its distance to a random subset of that population. Let $S(P) \subset P$ be a random subset of the population. We can then write

Taken from original publication: Thomas Gabor, Lenz Belzner, and Claudia Linnhoff-Popien. Inheritance-based diversity measures for explicit convergence control in evolutionary algorithms. In *The Genetic and Evolutionary Computation Conference (GECCO)*, 2018

the distance-based fitness function as

$$f(x,P) = g(x) + \lambda * \sum_{x' \in S(P)} \frac{d(x,x')}{|S(P)|}. \quad (3)$$

We found that selections with $|S(P)| = 5$ already performed well enough so that little difference from evaluation against the whole population could be found. Due to this fact, we employ this method of randomization whenever possible for the experiments described in this paper.

Inherited Fitness. Inherited fitness allows the fitness of an individual to be influenced by the fitness of its ancestors [3]. Alongside large parts of its genome, any individual generated via mutation or recombination thus inherits (an approximation of) its parents fitness value. Formally, we can write

$$f(x,P) = (1-\kappa) * g(x) + \kappa * h(x) \quad (4)$$

where

$$h(x) = \begin{cases} f(x') & \text{if } x \text{ is a mutation of } x', \\ \frac{f(x_1)+f(x_2)}{2} & \text{if } x \text{ is a recombination of } x_1 \text{ and } x_2, \\ 0 & \text{otherwise,} \end{cases}$$

with $\kappa \in \mathbb{R}, 0 < \kappa < 1$ being a *relative* weighting factor of the inherited fitness versus the currently computed objective value.

This approach attempts to aid diversity by slowing down the process of convergence: Even when individuals with very high objective goal values are discovered, it takes them a few generations to reach their full potential fitness value. This gives individuals in other niches more time to perhaps discover competitive solution candidates. In contrast to the individual-based approaches we introduced previously, inherited fitness does not introduce additional dependencies on the problem domain but operates solely on the fitness values generated with the help of the objective goal function that is given anyway. However, we recognize that the concept of diversity induced by this model is relatively weak since the combined fitness function does not depend on other peers in the population but only on the respective individual's parents.

Exact Genealogical Diversity. In an attempt to combine the benefits of distance-based diversity and inherited fitness, we introduce the notion of genealogical diversity [10]. In the end, we want to achieve a functional metric of the diversity of a single individual with respect to its current population without depending on any additional domain-specific knowledge such as distance functions. Instead, we want to use the knowledge already generated by the evolutionary process to give an estimate of the diversity of single individuals. This knowledge mainly stems from the application of evolutionary operators, i.e., it contains the genealogical relations of each individual. The ulterior idea behind this approach is that we can estimate that individuals that are closely related are less likely to be diverse with respect to each other. In short, you likely are more different from your cousin twice-removed than from your child.

We annotate every individual with a map $G : D \times D \to \mathbb{R}$ containing its genealogical distance to each other individual in the population. Whenever we generate an individual, it inherits the list of its parents and updates it accordingly, i.e., assuming x results from the recombination of x_1 and x_2 we can assign to x

$$G(x,x') := \begin{cases} 0 & \text{if } x' = x \\ r + \min\{G(x_1,x'), G(x_2,x')\} & \text{otherwise,} \end{cases}$$

where r is a parameter describing the distance we value a parent-child relationship with (usually $r = 1$). A similar definition can be made for children produced by mutation. Note that when individuals x and x' are completely unrelated, for example when one of them was newly generated by hypermutation, we assign some maximum value t which we also divide the results of G by in order to achieve normalization again. The resulting fitness function then looks pretty standard as

$$f(x,P) = g(x) + \lambda * \frac{\sum_{x' \in S(P)} G(x,x')}{|S(P)| * t}. \quad (5)$$

Genealogical Fitness. When directly applying exact genealogical fitness as described above, we again run into a complexity issue: We need to save the information of the whole tree of genealogical relations produced by the evolutionary process. Even if we limit ourselves to the distances between individuals still present in the population, we end up with a spatial complexity $O(|P|^2)$. For the populations used for the experiments, this was manageable and we thus performed these experiments for exact genealogical fitness as well. However, we still want an approach that scales well even with much larger populations. To this end, we were inspired by the way researches in biology determine the relatedness between singular individuals: They match their genomes. Compared to biological systems artificial evolution usually features much smaller genomes. Furthermore, large part of biological genomes are actually not subjected to selection pressure and can thus record patterns (and by extent genealogical relationships) without bias. We try to mimic these properties for our final approach towards diversity-aware evolutionary algorithms.

In order to efficiently approximate the genetic relation between two given individuals (without keeping a complete history of the whole evolutionary process), we assign every individual a bitstring $b = b_0, ..., b_{\tau-1}$ of length τ. Note that τ is the only hyperparameter introduced by genealogical diversity (and we noticed to be very robust with respect to different choices of τ). Every time an operator like mutation or crossover is applied to some individuals, we apply the respective operator to their assigned bitstrings. Since bitstrings are a classic among the representations used in genetic algorithms, most operators (even when designed for other data structures) have an immediate counterpart defined on $\{0,1\}^*$, which is the alphabet of bitstrings. Whenever an individual is newly created, it is assigned a random bitstring. When we choose τ large enough, these random bitstrings will feature a relatively high Hamming distance to each other. Throughout the process of evolution, we will interpret a high Hamming distance between two individuals as a sign of non-relatedness. Note that the use of the Hamming distance here does not depend on the problem domain but only on our choice to use bitstrings to augment the problem-specific genomes. More specifically, genealogical diversity is defined via a genealogical distance function $d : P \times P \to \mathbb{N}$, which can be written

Taken from original publication: Thomas Gabor, Lenz Belzner, and Claudia Linnhoff-Popien. Inheritance-based diversity measures for explicit convergence control in evolutionary algorithms. In *The Genetic and Evolutionary Computation Conference (GECCO)*, 2018

as

$$d(x,x') = \frac{1}{\tau} * \sum_{i=0}^{\tau-1} \begin{cases} 0 & if\, b_i = b'_i \\ 1 & otherwise \end{cases}$$

where $b_0, ..., b_{\tau-1}$ is the bitstring assigned to x and $b'_0, ..., b'_{\tau-1}$ is the bitstring assigned to x'. We then again can simply write

$$f(x,P) = g(x) + \lambda * \sum_{x' \in P} \frac{d(x,x')}{|S(P)|} \tag{6}$$

for the genealogically diverse fitness function. During the course of evolution, the bitstrings are not subject to selection but are subjected to the variational evolutionary operators. Thus, the bitstring can record the degree of relatedness between individuals, albeit in a highly probabilistic manner. Still, evolutionary algorithms show robustness with respect to the added noise in selection and comparing a few bitstrings to estimate diversity is highly efficient in both time and space complexity (with $\tau \approx log(|P|)$ being a good setting from an experimenter's experience).

4 EXPERIMENTAL RESULTS

To analyze the effectiveness of the various forms of diversity-respecting evolutionary algorithms described in this paper, we have considered various settings. In Subsection 4.1 we first describe a common benchmark problem often used to evaluate evolutionary algorithms [1]. Subsection 4.2 describes the setup and evaluation of a navigational problem written as a real vector optimization. Finally, Subsection 4.3 considers an integer combinatory problem.

4.1 The Schwefel Problem

The Schwefel problem has often been used as a benchmark problem for evolutionary algorithms [6, 11]. For this experiment, we used the implementation given by the DEAP library [1, 9]. The Schwefel problem is parametrized on the dimension of the search space (we simply write $|D|$) and given as the function

$$g(x) = 418.9828872724339 * |D| - \sum_{i=1}^{|D|} x_i * \sin(\sqrt{|x_i|}) \tag{7}$$

with the optimal solution to the minimization problem being $(0)^{|D|}$. Figure 1 provides a visualization of the solution landscape.

Setup. We show an evaluation of the Schwefel problem with $|D| = 8$. For all evolutionary algorithms, we used a total population size of 30 and ran evolution for 300 generations. For all experiments, we used a single-spot mutation operator and uniform recombination. We consistently chose relatively high values for variational parameters by opting for a mutation rate of 0.1, a recombination rate of 0.3, and a hypermutation rate of 0.1. We did so to have all algorithms benefit from diversity through increased randomness in the evolutionary process and thus evaluate their ability to produce diversity beyond adding random noise. For fitness sharing, we set $\alpha = 2$ and σ to the maximum value, so that it spans the whole problem domain. Inherited fitness used $\kappa = 0.2$. Genealogical diverse algorithms used a bitstring size of $\tau = 16$. The ensemble approach split the population into 3 subpopulations with a random migration rate of 0.1. For all weighted diversity mechanisms (those featuring a weighting factor λ in their fitness function) we chose $\lambda = 200$ for the Schwefel experiment. All parameters were approximated for best performance via manual grid search. For all experiments performed for this paper, we tested two variants of the weighted diversity mechanisms. In one case, we used the diversity term as described throughout this paper, which means it adds a bonus value in the direction of the optimization process. We also evaluated variants that (instead of rewarding high diversity values) penalize low diversity values by moving the respective individuals away from the optimization goal. Unsurprisingly, no real difference was observed in this regard.

Results. Due to the high amount of local optima it proved difficult for all tested algorithms to find the global optimum. The results in Figure 2 show that none of the algorithms reached the minimum

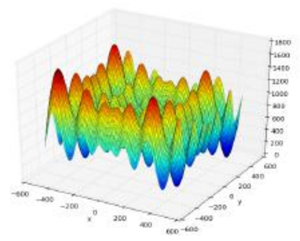


Figure 1: Illustration of the Schwefel problem in two dimensions. Taken from [1].

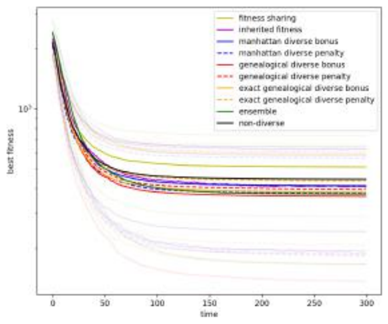


Figure 2: Evaluation results for the Schwefel problem. For each generation, we plot the current population's best objective value on a log scale. Averaged over 100 independent runs. Semi-transparent lines show plus/minus one standard deviation.

Taken from original publication: Thomas Gabor, Lenz Belzner, and Claudia Linnhoff-Popien. Inheritance-based diversity measures for explicit convergence control in evolutionary algorithms. In *The Genetic and Evolutionary Computation Conference (GECCO)*, 2018

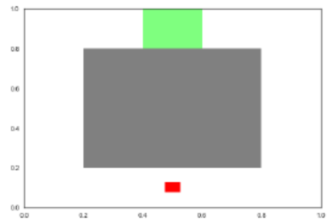


Figure 3: Illustration of the Pathfinding problem. A robot (red) gets rewarded for each of the 10 time steps of its life that it spends in the target area (green). It thus needs to reach the goal quickly using steps of size 0.3 in each dimension.

in the given time. While all algorithms perform extremely similar, the genealogical variant can be seen at the lowest point, although without too significant of a difference.

We ran this experiment for other benchmark problems contained in DEAP, notably H1, Schaffer and Rosenbrock [1, 6]. But the difference between various algorithms was even smaller for these experiments, which is why we left out the respective plots. All software and results are available online.[1]

4.2 The Pathfinding Problem

Given a room of dimensions 1×1, we imagine a robot standing at position $(0.5, 0.1)$. It needs to reach a goal area at the opposite side of the room, given by the square of side length 0.2 centered around the point $(0.5, 0.9)$. However, the room features a huge obstacle between those points and thus the robot needs to decide on a way around it. The agent can move by performing an action $a \in \{(\delta x, \delta y) | -0.3 < \delta x < 0.3, -0.3 < \delta y < 0.3\}$. The robot needs to develop a plan consisting of 10 such actions that will get it to the goal area. Once it has reached that area, it gets rewarded for staying there as long as possible. This setup is illustrated in Figure 3.

Setup. The Pathfinding problem has a dimension of $|D| = 20$ with $D = (\delta x_1, \delta y_1, \delta x_2, \delta y_2, ..., \delta x_{10}, \delta y_{10})$. The robot earns a reward of 1 for each time step spent within the goal area and receives a penalty of -0.1 for each attempt to perform an illegal step, i.e., a step ending up outside the room or inside the obstacle. Illegal steps are disregarded entirely (so the robot does not move *up to* the wall when attempting to step beyond it).

For the experiment, we used a population size of 100 individuals for all evolutionary algorithms. We allowed them to run for 1000 generations. Again, we used $\alpha = 2$ and $\sigma = \max$ for fitness sharing, split the population into 3 subpopulations for the ensemble method (yielding 34, 33, 33 for the subpopulation sizes), and set $\tau = 16$ for the genealogical algorithm. For all weighted diversity mechanisms we used $\lambda = 12$ this time, putting a high stress on diversity.

Results. For the Pathfinding problem, favoring diversity pays off in the optimization result. The results are shown in Figure 4. Distance-based diversity in the form of Manhattan diversity performed best. Genealogical diversity is a close second, however, achieving similar levels of results without a domain-specific distance function. On third place, fitness sharing too reaches similar results but is computationally more expensive. Using exact genealogical diversity, which was the original motivation for genealogical diversity, seems to have little effect on the result. We argue that parameters like the relative weight of recombination and mutation relations require further tweaking towards the problem-specific requirements. We find, however, that this defeats the original purpose of employing inheritance-based diversity measures in the first place. We thus focus on the results of the bitstring-augmented genealogical diversity instead. On a surprising note, both inherited fitness and the ensemble model perform worse than the standard evolutionary algorithm. Both may show a slowing effect on the evolutionary process. This means that for this problem, different random initialization of subpopulation most likely plays no role in enhancing diversity as even remotely competitive solution candidates are only found later on. It thus seems that these methods may in fact tackle different classes of problems.

4.3 The Routing Problem

For the last experiment, we wanted to opt for a discrete combinatorial problem in contrast to the continuous optimization of real-valued vectors performed so far. We again imagined a robot as it may work in a smart factory in the near future. This time, the robot already knows how to best travel to any given target, maybe involving various means of transport like forming convoys of robots or using conveyor belts installed in the factory. The robot is given the task to travel to various workstations that exist inside the factory in order to process a specific item it is carrying around. This item needs a certain amount of tasks to be performed in order to be produced. For each of these tasks, there are 5 dedicated workstations scattered throughout the factory. Figure 5 shows a smaller instance of that setup.

Setup. For this experiment, we choose a setting with 12 different tasks, resulting in a factory with 60 workstations. Accordingly, solution candidates are of the type $\{1, 2, ..., 5\}^{12}$. The genome $(2, 4, ...)$, e.g., ist interpreted as "go to the workstation of type A with the number 2; go to the workstation of type B with the number 4; ..." so that no type mismatch can ever happen. To mimic various means of transport, we randomized the distance between each of these workstations individually within a range of $[0, 100] \subset \mathbb{R}$. Note that this (most likely) gives rise to a non-euclidean space the robot is navigating, making the problem as difficult as finding the shortest weighted path in an arbitrary tree.

For the parameters of evolution, we used a population of 50 and ran each algorithm for 100 generations. Fitness sharing kept $\alpha = 2$ and $\sigma = \max$ but employed the Hamming distance instead of Manhattan distance to compute the niching radii. Inherited and genealogical fitness kept $\kappa = 0.2$ and $\tau = 16$, respectively. Ensemble model again used 3 subpopulations. For distance-based diversity, the Manhattan distance is no longer applicable since there is no associated meaning with the numbering of the workstations of a specific type. In this case, we can simply use Hamming distance

[1] gitlab.lrz.de/thomasgabor/gecco-evolib

Taken from original publication: Thomas Gabor, Lenz Belzner, and Claudia Linnhoff-Popien. Inheritance-based diversity measures for explicit convergence control in evolutionary algorithms. In *The Genetic and Evolutionary Computation Conference (GECCO)*, 2018

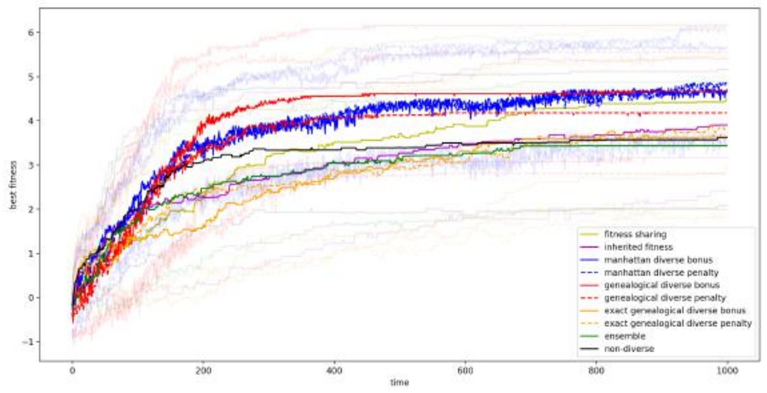


Figure 4: Evaluation results for the Pathfinding problem. For each generation, we plot the current population's best objective value. Averaged over 20 independent runs. Semi-transparent lines show plus/minus one standard deviation.

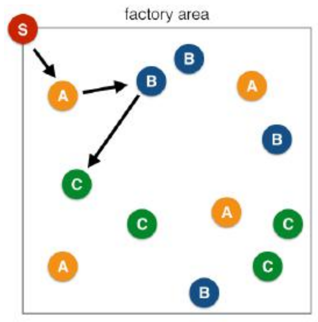


Figure 5: Illustration of the Routing problem. A robot *S* (red) needs to drive to workstations of types A, B, C in order. For each workstation type, it can choose from several workstations inside the factory. We need to find the routing path that minimizes the travelled distance.

instead. But it shows that for more complicated genome types, additional engineering effort may be necessary here. For all weighted diversity mechanisms we set $\lambda = 250$.

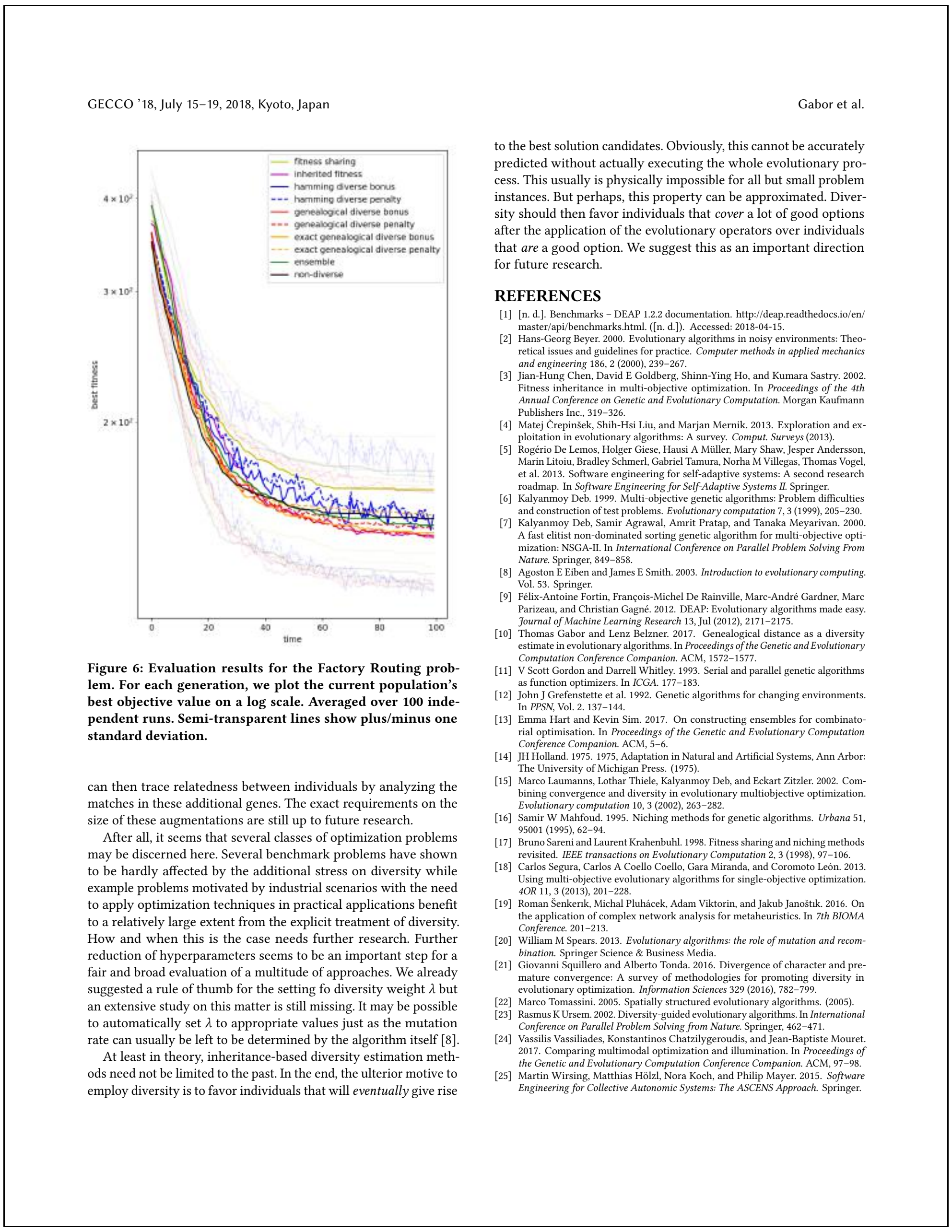
Results. The results of the factory routing experiment are depicted in Figure 6. We observe that fitness sharing seems to not perform as well using the Hamming distance function. The exact genealogical diversity approach again seems to make not that much of a difference. In this setting, so does the ensemble approach. However, the other three means to establish diversity do manage to achieve slightly better results. They all perform on a comparable level in the best cases, with Manhattan-based diversity being subject to more fluctuation than inherited or genealogical fitness.

5 CONCLUSION

While diversity has been known to be an important factor for the analysis of evolutionary algorithms, we focused on the explicit integration of diversity into a single-objective fitness function, which is a method not yet fully explored. Connections of this approach to standard multi-objective evolutionary algorithms and respective approaches that add diversity as an additional full-fledged objective are still to be researched. Furthermore, we focused on the issue of global optimization, i.e., we evaluated the tested algorithms for their ability to better approximate the global optimum only instead of, e.g., achieving a better coverage of various local optima [24]. We tested the approach of explicit weighted integration against common diversity techniques like fitness sharing, ensemble evolutionary algorithms or inherited fitness.

While domain-specific distance functions have been evaluated to be the most successful in several examples, we also aimed to provide means of measuring diversity that can more simply be plugged into existing algorithms (and libraries) without requiring as much domain-specific adjustments. For this purpose, we motivated and introduced the novel approach of genealogical diversity for a full evaluation. Inspired by nature, this approach augments the genomes by data structures not subjected to selection bias. We

Taken from original publication: Thomas Gabor, Lenz Belzner, and Claudia Linnhoff-Popien. Inheritance-based diversity measures for explicit convergence control in evolutionary algorithms. In *The Genetic and Evolutionary Computation Conference (GECCO)*, 2018



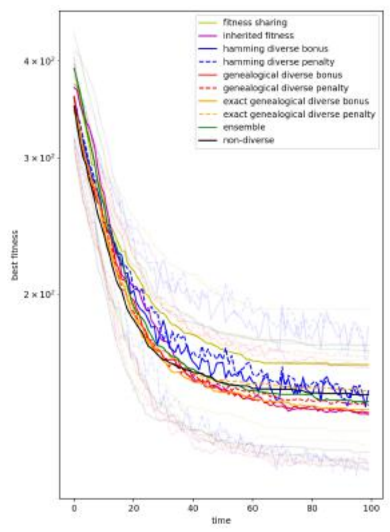


Figure 6: Evaluation results for the Factory Routing problem. For each generation, we plot the current population's best objective value on a log scale. Averaged over 100 independent runs. Semi-transparent lines show plus/minus one standard deviation.

can then trace relatedness between individuals by analyzing the matches in these additional genes. The exact requirements on the size of these augmentations are still up to future research.

After all, it seems that several classes of optimization problems may be discerned here. Several benchmark problems have shown to be hardly affected by the additional stress on diversity while example problems motivated by industrial scenarios with the need to apply optimization techniques in practical applications benefit to a relatively large extent from the explicit treatment of diversity. How and when this is the case needs further research. Further reduction of hyperparameters seems to be an important step for a fair and broad evaluation of a multitude of approaches. We already suggested a rule of thumb for the setting fo diversity weight λ but an extensive study on this matter is still missing. It may be possible to automatically set λ to appropriate values just as the mutation rate can usually be left to be determined by the algorithm itself [8].

At least in theory, inheritance-based diversity estimation methods need not be limited to the past. In the end, the ulterior motive to employ diversity is to favor individuals that will *eventually* give rise to the best solution candidates. Obviously, this cannot be accurately predicted without actually executing the whole evolutionary process. This usually is physically impossible for all but small problem instances. But perhaps, this property can be approximated. Diversity should then favor individuals that *cover* a lot of good options after the application of the evolutionary operators over individuals that *are* a good option. We suggest this as an important direction for future research.

REFERENCES

[1] [n. d.]. Benchmarks – DEAP 1.2.2 documentation. http://deap.readthedocs.io/en/master/api/benchmarks.html. ([n. d.]). Accessed: 2018-04-15.
[2] Hans-Georg Beyer. 2000. Evolutionary algorithms in noisy environments: Theoretical issues and guidelines for practice. *Computer methods in applied mechanics and engineering* 186, 2 (2000), 239–267.
[3] Jian-Hung Chen, David E Goldberg, Shinn-Ying Ho, and Kumara Sastry. 2002. Fitness inheritance in multi-objective optimization. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*. Morgan Kaufmann Publishers Inc., 319–326.
[4] Matej Črepinšek, Shih-Hsi Liu, and Marjan Mernik. 2013. Exploration and exploitation in evolutionary algorithms: A survey. *Comput. Surveys* (2013).
[5] Rogério De Lemos, Holger Giese, Hausi A Müller, Mary Shaw, Jesper Andersson, Marin Litoiu, Bradley Schmerl, Gabriel Tamura, Norha M Villegas, Thomas Vogel, et al. 2013. Software engineering for self-adaptive systems: A second research roadmap. In *Software Engineering for Self-Adaptive Systems II*. Springer.
[6] Kalyanmoy Deb. 1999. Multi-objective genetic algorithms: Problem difficulties and construction of test problems. *Evolutionary computation* 7, 3 (1999), 205–230.
[7] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and Tanaka Meyarivan. 2000. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In *International Conference on Parallel Problem Solving From Nature*. Springer, 849–858.
[8] Agoston E Eiben and James E Smith. 2003. *Introduction to evolutionary computing*. Vol. 53. Springer.
[9] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. 2012. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research* 13, Jul (2012), 2171–2175.
[10] Thomas Gabor and Lenz Belzner. 2017. Genealogical distance as a diversity estimate in evolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM, 1572–1577.
[11] V Scott Gordon and Darrell Whitley. 1993. Serial and parallel genetic algorithms as function optimizers. In *ICGA*. 177–183.
[12] John J Grefenstette et al. 1992. Genetic algorithms for changing environments. In *PPSN*, Vol. 2. 137–144.
[13] Emma Hart and Kevin Sim. 2017. On constructing ensembles for combinatorial optimisation. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM, 5–6.
[14] JH Holland. 1975. 1975, Adaptation in Natural and Artificial Systems, Ann Arbor: The University of Michigan Press. (1975).
[15] Marco Laumanns, Lothar Thiele, Kalyanmoy Deb, and Eckart Zitzler. 2002. Combining convergence and diversity in evolutionary multiobjective optimization. *Evolutionary computation* 10, 3 (2002), 263–282.
[16] Samir W Mahfoud. 1995. Niching methods for genetic algorithms. *Urbana* 51, 95001 (1995), 62–94.
[17] Bruno Sareni and Laurent Krahenbuhl. 1998. Fitness sharing and niching methods revisited. *IEEE transactions on Evolutionary Computation* 2, 3 (1998), 97–106.
[18] Carlos Segura, Carlos A Coello Coello, Gara Miranda, and Coromoto León. 2013. Using multi-objective evolutionary algorithms for single-objective optimization. *4OR* 11, 3 (2013), 201–228.
[19] Roman Šenkerık, Michal Pluhácek, Adam Viktorin, and Jakub Janoštık. 2016. On the application of complex network analysis for metaheuristics. In *7th BIOMA Conference*. 201–213.
[20] William M Spears. 2013. *Evolutionary algorithms: the role of mutation and recombination*. Springer Science & Business Media.
[21] Giovanni Squillero and Alberto Tonda. 2016. Divergence of character and premature convergence: A survey of methodologies for promoting diversity in evolutionary optimization. *Information Sciences* 329 (2016), 782–799.
[22] Marco Tomassini. 2005. Spatially structured evolutionary algorithms. (2005).
[23] Rasmus K Ursem. 2002. Diversity-guided evolutionary algorithms. In *International Conference on Parallel Problem Solving from Nature*. Springer, 462–471.
[24] Vassilis Vassiliades, Konstantinos Chatzilygeroudis, and Jean-Baptiste Mouret. 2017. Comparing multimodal optimization and illumination. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM, 97–98.
[25] Martin Wirsing, Matthias Hölzl, Nora Koch, and Philip Mayer. 2015. *Software Engineering for Collective Autonomic Systems: The ASCENS Approach*. Springer.

Taken from original publication: Thomas Gabor, Lenz Belzner, Thomy Phan, and Kyrill Schmid. Preparing for the unexpected: Diversity improves planning resilience in evolutionary algorithms. In *15th IEEE International Conference on Autonomic Computing (ICAC)*, 2018

Preparing for the Unexpected: Diversity Improves Planning Resilience in Evolutionary Algorithms

Thomas Gabor
LMU Munich
thomas.gabor@ifi.lmu.de

Lenz Belzner
MaibornWolff
lenz.belzner@maibornwolff.de

Thomy Phan
LMU Munich
thomy.phan@ifi.lmu.de

Kyrill Schmid
LMU Munich
kyrill.schmid@ifi.lmu.de

***Abstract*—As automatic optimization techniques find their way into industrial applications, the behavior of many complex systems is determined by some form of planner picking the right actions to optimize a given objective function. In many cases, the mapping of plans to objective reward may change due to unforeseen events or circumstances in the real world. In those cases, the planner usually needs some additional effort to adjust to the changed situation and reach its previous level of performance. Whenever we still need to continue polling the planner even during re-planning, it oftentimes exhibits severely lacking performance. In order to improve the planner's resilience to unforeseen change, we argue that maintaining a certain level of diversity amongst the considered plans at all times should be added to the planner's objective. Effectively, we encourage the planner to keep alternative plans to its currently best solution. As an example case, we implement a diversity-aware genetic algorithm using two different metrics for diversity (differing in their generality) and show that the blow in performance due to unexpected change can be severely lessened in the average case. We also analyze the parameter settings necessary for these techniques in order to gain an intuition how they can be incorporated into larger frameworks or process models for software and systems engineering.**

***Index Terms*—planning, unexpected events, dynamic fitness, resilience, robustness, self-protection, self-healing, diversity, optimization, evolutionary algorithms**

I. Introduction

As automatic optimization in various forms makes its way into industrial systems, there is a wide range of expectations about the upcoming capabilities of future "smart systems" [1]–[5]. For most of the current applications, the optimization part of the system takes place *offline*, i.e., not while the application is actually performing its main purpose: The product shipped to the customer is fixed after initial training and does not self-adapt (anymore). Instead, it may only gather data that is then used at the vendor's side to either improve the product's performance via software updates later on or assist in building the product's successor. This, of course, misses out on interesting applications that may highly benefit from further optimization even while they are running. In this paper, we focus on the exemplary case of a layout configuration for the positioning of work stations inside a (smart) factory: Depending on the products that need to be build and depending on the current status of the machines involved, we may desire different workflows for the same product at different times during the factory's life. For most current factories, however, the arrangement of workstations is planned far in advance and then fixed until human intervention.

One of the reasons for opting for offline adaptation is that the vendor usually has access to more computational power and that the employed adaptation process can benefit from connecting data input from a variety of customers. However, increasing computational resources and online connectivity mitigate these issues. A possibly more important aspect is the issue of consistent performance: An online planner, while theoretically able to react to sudden changes in its environment and/or objective, may take some time to reach good plans and during that time the solutions provided by the planner may be unsuitable.

a) Expected Change: The usefulness and importance of *self-optimization* at the customer's side has already been claimed in the original vision of autonomic computing [6] and has been shown on many occasions since [3], [7], [8]. In these cases, self-optimization usually refers to a process of specialization, i.e., the system is built with a large variety of possible use cases in mind and learns to work best for the few of these it actually faces on site. Intuitively, we may want to build a planner that works on factory layouts in general and that can then specialize on the specific needs of a single factory or a single situation (machine failure, e.g.) if necessary. We expect this approach to work iff every possible situation and every pair of follow-up situation is considered when evaluating a factory layout. As long as we know that machines might fail with a certain probability, we can take this into account and plan redundantly with respect to machine usage. This is what we call *expected change* of the evaluation function.

b) Unexpected Change: Still, we may not want our self-optimizing planner to completely break on any deviation from the specified scenarios. We imagine that intelligent planners should invest a certain amount of effort to think about and prepare for "what ifs", even when the respective scenarios have not been expected to happen during system design or training. This is further motivated by the fact that many industry applications require the adaptive component to produce a

Taken from original publication: Thomas Gabor, Lenz Belzner, Thomy Phan, and Kyrill Schmid. Preparing for the unexpected: Diversity improves planning resilience in evolutionary algorithms. In *15th IEEE International Conference on Autonomic Computing (ICAC)*, 2018

solution better than a certain quality threshold but do not benefit as much from the system finding configurations that are just slightly better beyond that threshold. Instead, that computational effort might be better put into finding alternative solutions that might not be just as good as the primary solution that was just found, but then again might be feasible even when the primary solution fails for some *unexpected* reason.

This argument falls in line with the claim of *self-protection* for autonomic systems [6]: Our system should not only be able to react and recover from negative external influences but also spend a reasonable effort on actively preparing for negative events. Via this self-protection property we aim to increase the overall resilience of the planning process and by extent the robustness of the system using our planner.

c) Scope of This Work: As the original contribution of this paper we identify that diversity in evolutionary algorithms, which we consider a primary example for a heuristic optimization algorithms in this paper, is of central importance for the algorithm's reaction to change and that explicitly optimizing for diversity helps to prepare for changes, even when they cannot be foreseen by the optimization process in any way. We introduce means to formally define the phenomenon of unexpected change in relation to an online planner.

To this end, we first formally define the notions of change and unexpectedness that we used intuitively until now (Section II). We then immediately turn to an example of a smart factory domain in which unexpected change might occur and specify our experimental setup (Section III). We introduce our approach at maintaining diversity using two different diversity metrics (Section IV) and sum up the results of applying this approach in the previously defined experiment (Section V) before we discuss related work (Section VI) and conclude this paper (Section VII).

II. Foundations

We assume that to realize modern challenges in industry, software products need to feature a certain degree of *autonomy*, i.e., they feature at least one component called *planner* capable of making decisions by providing a plan of actions which the system is supposed to perform to best fulfill its intended goal [8], [9]. This goal is encoded by providing the system with a *fitness function* that can be used to evaluate plans. A planner respecting a fitness function performs self-optimization.

We claim that for many real-world applications it is often not only important to eventually adapt to new circumstances but also to avoid causing any major damage to overall success while adapting. It follows that the planner needs to offer a suitable solution at all times, even directly after change in the environment. This property can be compared to the *robustness* of classical systems, i.e., the ability to withstand external changes without being steered away too far from good behavior [10]. Robustness can often be tested against a variety of well-defined external influences. However, not every influence a system will be exposed to can be foreseen.[1] The notion of *resilience* captures the system's ability to withstand *unanticipated* changes [11].[2] One approach to prepare a system for unexpected circumstances is to make it adapt faster, so that its adaptive component finds a new plan of actions faster once the old one is invalidated. However, this approach is still purely reactive and we thus cannot prevent the immediate impact of change.

To increase system resilience, we thus might want the planner to become proactive towards possible changes that may occur to the environment and by extension the planner's objective. In order to lessen the blow of unexpected changes, the planner thus needs to prepare for it before it actually occurs. Note that for the changes we are talking about in this section, we still assume that they are unexpected at design time. The planner therefore has no means of predicting when or what is going to happen. Still, we desire for a planner to be caught off-guard as seldom as possible. A planner that needs to re-plan less often would then be considered more resilient with respect to unexpected change. We claim that explicitly increasing planning resilience aids a system's ability to self-protect and is thus a useful handle to explicitly expose to the developers of such a system.

a) Planning: Planners perform (usually stochastic) optimization on the system's behavior by finding plans that (when executed) yield increasingly better results with respect to a specified objective. That objective is given via a fitness function $f : P \times E \to \mathbb{R}$, where P is the domain of all possible plans and E is the domain of environments said plans are to be executed in. For the purpose of this paper, we assume that we want to *minimize* the real-valued output of the fitness function. We can then describe a planner formally as a function *plan* $: E \to P$ from an environment $e \in E$ to a plan $p \in P$ with the following semantic:

$$\mathit{plan}(e) \approx \arg\min_{p \in P} \mathbb{E}(f(p, e)).$$

Note that due to the possibly stochastic nature of the environment and in extent the evaluation of the fitness function f, we compute the expected value $\mathbb{E}$ of the application of f. Further note that due to the stochastic nature of the planning methods considered in this paper, we may not actually return the single best result over the domain of all plans but when the stochastic optimization process works, we expect to yield a result somewhat close (described by $\approx$). To compute a reasonable value for $f(p, e)$, a given plan will usually be executed in a simulated version of e. We call the process of repeatedly calling *plan* to execute the currently best solution *online* planning, which implies that we may call it for changing e.

[1]When possible, endowing systems with means to perceive all possible influences and events might be highly beneficial to resilience. We work with the assumption that this is not always possible or feasible.

[2]It follows that we consider resilience a special instance of robustness: Robustness may include both anticipated and unanticipated change. Resilience focuses on the latter.

Taken from original publication: Thomas Gabor, Lenz Belzner, Thomy Phan, and Kyrill Schmid. Preparing for the unexpected: Diversity improves planning resilience in evolutionary algorithms. In *15th IEEE International Conference on Autonomic Computing (ICAC)*, 2018

b) Changing Environments: We can write any occurrence of change in the environment as a function $c : E \to E$. Obviously, if we allow any arbitrary change to happen to the environment, we can construct arbitrarily "evil" environments and cause the planner to perform arbitrarily bad. But frankly, we do not care for a planner managing a smart grid's power production to perform well when a meteor destroys Earth. What is much more realistic and thus much more desirable to prepare for, however, is changes that apply only to parts of the environment. Without looking into the data structure of the environment, we assume that these kinds of changes then only affect the fitness of some possible plans, but do not change the fitness landscape of the domain completely. We thus call a given change function c within a given environment $e \in E$ *reasonable* iff it fulfills the formula:

$$|\{p \in P : |f(p,e) - f(p,c(e))| > \varepsilon\}| \ll |P|.$$

Here, ε described a small value used as a minimally discernible distance between fitness values. Likewise, the exact meaning of $\ll$ is to be defined by the case. From this definition, it follows that a planner can prepare for a reasonable change by finding a good plan among the plans that are not affected by the reasonable change. When the change occurs, it can then provide a "quite good" plan immediately before it even begins to search for good plans among the changed parts of the domain. Thus, to increase planning resilience, we want our planner to not converge on and around the best optimum it has found so far, but to always keep an eye out for other local optima, even when they do not look as promising at the moment.

Note that this behavior can be likened to strategies developed to prevent premature convergence, a problem with metaheuristic search methods that occurs even in static domains [12], [13].

c) Unexpectedness: Even if a planner can prepare for a reasonable change by diversifying, there are often more efficient ways to prepare for expected change: Usually, we would include instances of expected change into the fitness function by simply evaluating the system in the changed environments as well. In that case, the planner can still fully converge on the predicted path of the environment and not spend computational resources on diversification. However, we claim that in most practical applications the future is not completely predictable and changes may happen that the planner cannot anticipate.

We define a change function c to be called *unexpected* iff the planner is not prepared for the change induced, i.e., if the actions it would take in the unchanged environment e differ from the actions it now has to take in the changed environment $c(e)$. Formally, this can be expressed as follows:

$$|\{e \in E : \mathit{plan}(c(e)) \not\approx \mathit{plan}(e)\}| \gg 0$$

Again, an exact definition of $\gg$ would need to be derived from specific system requirements. Note that this is a purely extrinsic view on unexpectedness. We want to provide a black-box definition of unexpectedness that does not depend on the internal workings of the planner and is thus as general as possible. The intuition behind it is that if there was a way for the planner to know that and how the change c is going to happen when looking at the environment e, the plan generated via $\mathit{plan}(e)$ would already consider the consequences of said change and thus (to some extent) match the plan for $c(e)$.[3]

III. Experiment

To test the validity of our claims about the importance of diversity for planning resilience, we build a model example in which we try to observe the effects of environmental changes as clearly as possible.

a) Scenario: We imagine a smart factory scenario where a work piece carried by a mobile (robotic) agent needs to be processed by a setup of work stations. More specifically, we need to perform the 5 tasks A, B, C, D, E in order on a given work piece as quickly as possible. In order to do so, our factory contains 25 work stations placed randomly on a 500×500 grid structure. Each work station can only perform one of the tasks, so that our factory has 5 identical work stations to use for any specific task. Given a work piece starting at the top left corner of the grid, we need to determine the shortest route the work piece can travel for it to reach exactly one station of each task in the right order. See Figure 1 for a simplified illustration of this setup.

For each run of our experiment, we randomly generate an $n \times m$ matrix F of work station coordinates where each row in F corresponds to a task and each column to an identification number for each of the available work stations for each task. Thus, in our experimental setup we fix $n = 5$ and $m = 5$.

b) Genetic Algorithm: In order to find a short path that fulfills our requirements, we employ a genetic algorithm [12]. Closely modeling our problem domain, we define the genome as a 5-dimensional vector $v \in \{0, ..., m-1\}^n$ so that v_i denotes which of the 5 identical work stations should be visited next in order to fulfill the i-th task where $i = 0$ denotes the task A, $i = 1$ denotes task B, and so on. The environment provides a mapping from these v_i to their respective positions on the grid, which is used by a distance function L^E for the environment E to compute the traveling distance between two work stations. We then define a function *waycost* to compute the overall length of a given path, summing the Manhattan[4] distance L_1^E between all its vertices:

$$\mathit{waycost}(v,E) = L_1^E(S, v_0) + \sum_{i=0}^{n-2} L_1^E(v_i, v_{i+1})$$

[3]Note that this argument is based on the fact that we defined *plan* in such a way that it tries to optimize for $f(p,e)$ when possible. The result is that we can regard the definitions of "reasonable" and "unexpected" as upper and lower bounds on the amount of change introduced in the fitness landscape.

[4]Obviously, real-world mobile transport robots are more likely to navigate in Euclidean space. However, we argue that this is not crucial for the results presented in this paper and choose the computationally simpler approach.

Taken from original publication: Thomas Gabor, Lenz Belzner, Thomy Phan, and Kyrill Schmid. Preparing for the unexpected: Diversity improves planning resilience in evolutionary algorithms. In *15th IEEE International Conference on Autonomic Computing (ICAC)*, 2018

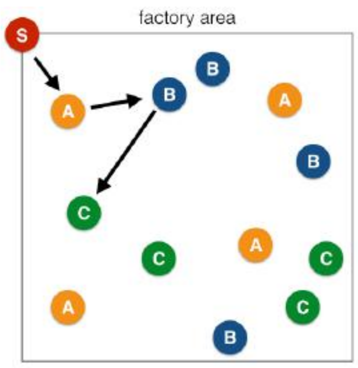


Fig. 1. Illustration of the factory setup, simplified for only 3 tasks A, B, C and 4 stations for each task. Coming from the starting point S, the genetic algorithm needs to determine an as short as possible path that traverses a station for each task in order (see arrows).

For the standard genetic algorithm, this *waycost* function is already sufficient as a fitness function $f(v, E) =$ *waycost*(v, E) to evolve a shorter navigation path. It is important to note that while we closely restrict the search space to paths that cross each type of station exactly once (and in the right order), we do *not* aid the genetic algorithm by providing any notion of position in space or the closeness of different stations beyond what is encoded in the *waycost* function above.

For the genome defined above, we use the following evolutionary operators: Mutation chooses a value $i, 0 \leq i < n$ uniformly at random, then generates a new random value $x \in \{0, ..., m-1\}$, assigning $v_i := x$. Recombination happens through uniform crossover on the vectors of two individuals. Furthermore, for all experiments performed in this paper, we use a mutation rate of 0.1 per individual to provide strong random input and a crossover rate of 0.3. That means that with a chance of 30% per individual that individual is selected as a first mate for recombination. Two potential mates are then randomly selected from the population: the fitter one is used for as a partner for crossover. We further augment the search by randomly generating some new individuals from scratch each generation. This process (also called hyper-mutation [14]) happens with a chance of 0.1 per individual in the population.

c) Random Change: The crucial point of this experimental setup is the occurrence of a random change of environmental circumstances. The present experimental setup is fixed to an evaluation time of 100 generations as earlier experiments have shown our setup of an evolutionary algorithm can easily converge in under 50 generations. We then define a function for unexpected change c_A, which chooses A factory stations at random and effectively disables them. This is implemented by repositioning them to an area far off the usual factory area by adding $(2500, 2500)$ to their respective coordinates. This means that while the plans containing the removed stations are still theoretically feasible and can be assigned a valid *waycost*, the increase in *waycost* is so high that no plan containing any of the removed stations should be able to compete with plans contained within the actual factory area when it comes to evolutionary selection. From a random initial factory layout F we generate two changed factory layouts $F_1 = c_A(F), F_2 = c_A(F)$ by applying the randomized change function c_A. Because we want to be able to compare the scale of fitness values before and after the unexpected change more easily, we start the evolutionary algorithm on the factory configuration F_1 that is already "missing" a few stations. After 50 generations, we switch to factory configuration F_2, which has A stations disabled as well, but probably different ones.[5]

Note that this change is reasonable for small A (according to the definition above) because it only affects the fitness of a maximum of $2 * A$ possible plans, i.e., those plans which include at least one of the "wrong" machines in F_1 or F_2. Furthermore, the change is unexpected as the shakeup of the stations' positioning is communicated to the evolutionary algorithm only via the change of the *waycost* function's values in its fitness evaluation step and thus leaves the adaptation process without any chance of anticipating that event. Nonetheless, the individuals of the evolutionary process are constantly evaluated according to their fitness in the current state of affairs, thus forcing them to adapt to the new situation in order to keep up once reached levels of fitness values.

IV. Approach

We attempt to solve the problem described above using evolutionary algorithms. Evolutionary algorithms have already been applied successfully to many instances of online adaptation, i.e., problems with a changing fitness function [15]–[17]. They are an instance of metaheuristic search algorithms and work by emulating natural evolution.

a) Diversity in Genetic Algorithms: In the standard scenario, once the fitness function changes, previously good solutions can possibly be evaluated to have very bad fitness and are thus removed from the evolutionary process. However, if the genetic search has already converged to a local optimum, it can be very hard for the search process to break out of it, because when all known solutions lie very closely together in the solution space, there is no clear path along which the population must travel in order to improve. The problem of

[5]It is important to note that this setup means that in many cases none of the stations that go bad during the switch are even included in the best path found by the genetic algorithm. In these cases, the evolutionary process does not have to adapt in any way. In order to analyze the cases when the removal of stations actually does make a huge difference, we need to execute the experiment multiple times. We chose this approach because it allows us use an unbiased change function as opposed to a change function that specifically targets the workstations actually used throughout the experiment. The realm of biased, even directly adversarial change functions is an interesting topic of future research.

Taken from original publication: Thomas Gabor, Lenz Belzner, Thomy Phan, and Kyrill Schmid. Preparing for the unexpected: Diversity improves planning resilience in evolutionary algorithms. In *15th IEEE International Conference on Autonomic Computing (ICAC)*, 2018

a genetic search getting stuck in a local optimum with little chance to reach the global optimum (or at least much better local ones) is called *premature convergence* [12]. It is known that the diversity among the members in the population has a strong impact on the evolutionary process's likelihood to converge too early. The Diversity-Guided Evolutionary Algorithm (DGEA) observes a population's diversity throughout the evolutionary process and takes action when it falls below a given threshold [18].

For online genetic algorithms, we show that maintaining a certain level of diversity throughout the population helps to react better to the change occurring in the environment. To this end, we apply two possible measurements for diversity, which we will both test for the above scenario. In either case, we transform the genetic algorithm's fitness function to a *multi-objective optimization* problem [13], [19], [20] with a weighting parameter λ, yielding a fitness function f depending on the individual to be evaluated v, the environment E, and the population P as a whole:

$$f(v, E, P) = \textit{waycost}(v, E) + \lambda * \textit{similaritycost}(v, P)$$

It is important to note that in order to meaningfully define the diversity of one individual, we need to compare it to the rest of the population, causing us to introduce the population P as an additional parameter to the fitness function.[6] The fitness function thus becomes a relative measure with respect to other individuals in the population. This makes it necessary to re-evaluate fitness in each generation even for unchanged individuals. However, since we assume changes in the environment and thus the fitness function may occur during the online execution of the genetic algorithm anyway, this model seems to fit our situation. We can now define two different diversity measures by providing a definition for the *similaritycost* function, which penalizes low diversity.

b) Domain-Distance Diversity: This can be thought of as the more standard approach to diversity in search and optimization problems. In fact, the authors of [22] show that many common diversity measurements are quite similar to this basic method: We define a simple distance measure between the individuals in the solution space. For a discrete, categorial problem like the one presented here, there is little alternative to just counting the specific differences in some way.

$$\textit{similaritycost}_{dom}(v, P) = -n + \sum_{i=0}^{n-1} \sum_{j=0}^{|P|} C(v_i, P(j)_i)$$

$$\textit{where } C(x, y) = \begin{cases} 1 & \textit{if} \quad x = y \\ 0 & \textit{otherwise} \end{cases}$$

[6]In general, we might want approximate this comparison by using a sample drawn from the population or another estimate instead. Likewise, we could consider computing diversity not only against the current generation of individuals but also against a selection of individuals from the past, using for example a "hall of fame" approach [21]. The evaluation of such techniques is left for future research.

Note that we write $P(j)$ to access the j-th individual of the population and $|P|$ to represent the amount of individuals in a population. We subtract n from the sum because the given individual $v \in P$ is still part of the population and thus adds a cost of n by matching itself perfectly. We thus maintain the (cosmetic) property that in a population of completely different individuals, the average similarity is 0.

While the implementation of this diversity measure looks pretty straightforward, it requires complete prior knowledge of the search space provided and thus introduces further dependencies. For example, the above definition is unfit for continuous search spaces and while a continuous *similaritycost* function may easily be thought up, optimization problems consisting of a mix of discrete and continuous variables then require more weighting parameters to adequately combine the scales over which the respective *similaritycost* functions operate.

c) Genealogical Diversity: As a more different comparison we implemented a inheritance-based diversity estimate introduced in [13]. The aim of genealogical diversity is to utilize those parts of the domain knowledge that are already encoded in the setup of the genetic algorithm, i.e., the mutation and recombination function the human developer is required to code for the specific genome anyway. We can thus try to quantify the difference between two individuals by estimating the amount of evolution steps it took to develop these different instances of solution candidates. This yields a measure of "relatedness" between individuals not unlike genealogical trees in biology or human ancestry. If all individuals in a population are closely related (sibling or cousins, e.g.), we know that there can only be limited genetic difference between them and thus estimate a low diversity for the respective individuals with respect to that population.

However, instead of building and traversing a genealogical tree, the implementation of genealogical diversity used in [13] employs a technique inspired by the way genetic genealogical trees are constructed from the analysis from genomes in biological applications: For this approach, we first need to augment the individuals' genome by a series of t trash bits $b_k \in \{0, 1\}, k \in \mathbb{N}, 0 \leq k < t$. For our experiment, $t = 16$ has proven to be reasonable. However, we do not change the *waycost* fitness function, so that it does not recognize the additional data added to the genome. This leads to the trash bits not being subjected to selection pressure from the primary objective of the genetic algorithm.

As the trash bits are randomly initialized like the other variables in the genome, every individual of the first generation should most probably start out with a very different trash bitstring from anyone else's, given that we choose the length of the trash bitstring sufficiently large. Without direct selection pressure, there is no incentive for individuals to adapt their trash bitstring in any specific way. However, the trash bits are still subjected to mutation and recombination, i.e., whenever a specific individual is chosen for mutation, a random mutation is performed on the trash bitstring as well and whenever a

Taken from original publication: Thomas Gabor, Lenz Belzner, Thomy Phan, and Kyrill Schmid. Preparing for the unexpected: Diversity improves planning resilience in evolutionary algorithms. In *15th IEEE International Conference on Autonomic Computing (ICAC)*, 2018

recombination operation is executed for two individuals, their trash bitstrings are likewise recombined. In our implementation at hand, we use one-bit flip for mutation and uniform crossover for recombination.

Using the definition of a comparison function C as provided above, we can thus define the *similaritycost* function for genealogical diversity as follows:

$$\textit{similaritycost}_{gen}(v, P) = -t + \sum_{i=0}^{t-1} \sum_{j=0}^{|P|} C(v_{n+i}, P(j)_{n+i})$$

Again, we subtract t to ignore self-similarity when iterating over the population. It should be noted that when accessing the $(n+i)$-th component of an individual inside the sum, we are protruding into the dimensions solely populated by trash bits, retrieving the i-th trash bit of said individual.

In order to compute the similarity between two individuals, we now only consider the trash bits, for which we always have the same distance metric regardless of the actual problem domain of the original genetic algorithm. Domain logic is only used indirectly, as the measure we estimate can be regarded as the edit distance between two individuals using the genetic operators the evolutionary process is equipped with. However, since the trash bits are inherited by individuals from their parents and without direct selection pressure, they are not biased toward values resulting in higher fitness; yet, they are still a sufficient representation of the genealogy of an individual, as we show in the following section.

V. Results

In order to evaluate the benefit of the presented approaches, we simulate the different behavior of genetic algorithms when using the presented diversity measures or no diversity measure at all. In order to achieve a meaningful result considering the highly probabilistic nature of the applied method to generate scenarios, we perform the evaluation on 1000 different scenarios. Figure 2 shows the top fitness achieved at a specific point in time by a single run averaged over all 1000 runs. By taking a look at the optimization process as a whole, it can be seen that a great deal of improvement compared to the random initialization is done during the first steps of evolution, giving an estimate of how good the achieved solutions are in relation to "just guessing". In Figure 3 we show the respective diversity measurements from these runs.

We can observe that the diversity-aware algorithms show a slower learning rate in the beginning, since they do not only optimize the plotted primary fitness function, but also the diversity function and thus cannot focus as well on better primary results. However, once the environmental change occurs, they are likewise better prepared for a change in fitness and react with a much smaller increase in *waycost* than the standard genetic algorithm. In a scenario like ours, where a smart factory needs to be able to efficiently dispatch new workpieces at all times, this can be a huge advantage. We observe that following the unexpected change, average diversity first increases as well-established "families" of similar individuals die out. Due to a new convergence process, diversity then drops until good solutions are found. Finally, diversity seems to reach a similar level as before the unexpected change. The "right" amount of diversity is naturally controlled by the parameter λ of the combined fitness function. For these

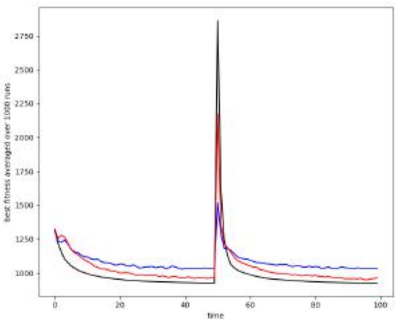


Fig. 2. Best (i.e., lowest-valued) fitness for current generation averaged over 1000 runs. While the evolutionary algorithm without any recognition of diversity (black) shows a steep spike at the time of the environmental change (after 50 generations), genealogically (red) and the domain-dependent (blue) diverse genetic algorithms manage to mitigate the negative amplitude to varying extent.

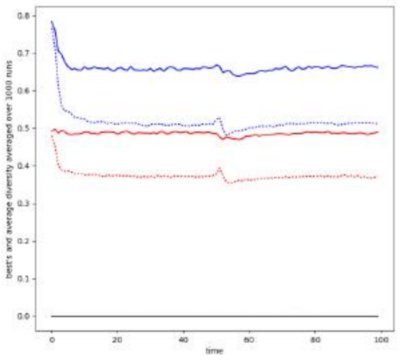


Fig. 3. Diversity measures of the top individual (solid line) as well as the population average diversity (dotted line) per generation averaged over 1000 runs. We draw both genealogical (red) and domain-dependent (blue) diversity into the same figure as they are both normalized on [0; 1], even though no direct translation is possible between their values. In both cases, the population's average diversity shows a specific behavior following the unexpected change.

Taken from original publication: Thomas Gabor, Lenz Belzner, Thomy Phan, and Kyrill Schmid. Preparing for the unexpected: Diversity improves planning resilience in evolutionary algorithms. In *15th IEEE International Conference on Autonomic Computing (ICAC)*, 2018

experiments we found parameters $\lambda = 1500$ for domain-dependent diversity and $\lambda = 2500$ for genealogical diversity via systematic search.

The definition of "right", however, depends on the problem domain. In most practical cases, we expect some (non-functional) requirements to be present, which specify the robustness properties we want to uphold. For now, these properties must then be verified via statistic testing. Deriving (statistical or hard) guarantees from a stochastic search process like an evolutionary algorithm is still an interesting topics of future work. Goven no further requirements for consistent quality of service, a reasonable setting for λ might achieve that the online planner does not perform worse than a random planner at any point in time, even at the moment of unexpected change.

Figures 4 and 5 show that systematic search, including the random population's value before the evolutionary process starts: the fitness achieved by the domain-dependent and the genealogical genetic algorithm, respectively, strongly depends on the choice of parameter λ, i.e., how to distribute focus between the primary objective (small *waycost*) and the secondary objective (high diversity). Experiments have shown, that diversity-aware genetic algorithms can show a variety of behaviors for different λ. To provide an intuition about the effects various settings for λ have on the algorithm's performance, we can see that higher values of λ generally cause the evolutionary search to produce less optimal results but to perform more stable when facing unexpected change. For the domain-dependent diversity, this phenomenon shows stronger with higher λ-values showing almost no impact of the unexpected change but relatively bad results in general. The approach of genealogical diversity seems to be a bit more robust to the setting of λ in that it still clearly shows a tendency to optimize over time.

We chose to showcase genealogical diversity specifically because it works on a rather domain-independent level and introduces only few parameters. Furthermore, it is rather robust with respect to the choice of said parameters. For the length of the used bitstring t, Figure 6 shows that on all but the smallest values for t the genetic algorithm performs most similarly. Especially rather large values for t (that still take up very little memory) do not show any deterioration in the planner's behavior, which means that the choice for that parameter can be made rather comfortably.

We also analyze *how much* change a diversity-aware planner can handle. Figure 7 shows the behavior of the three exemplary planners just around the moment of unexpected change for various amounts of change they are subjected to. Naturally, bigger (and thus un-reasonable) change can impact even diverse system. The increase in costs for the large alterations in the generation-49-line (dashed) shows that on the upper end of the scale we started generating problem instances that generally have fewer good solutions. For more reasonable change ($A \leq 8$, which still means that up to 16 out of 25 machine positions may be changed), both diversity-aware algorithms perform comparably and clearly better than the

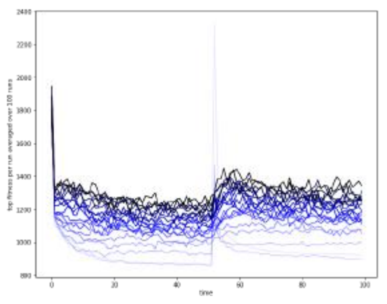


Fig. 4. Top fitness for current generation averaged over only 100 runs each, plotted for $\lambda = 500 * z, z \in \mathbb{N}, 0 \leq z < 20$ using domain-dependent diversity. The darker the color of the line, the higher is the depicted λ value.

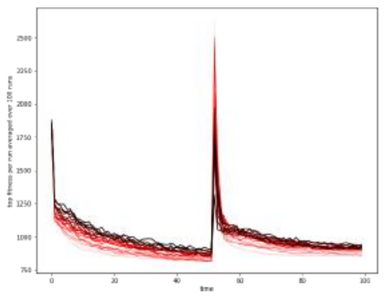


Fig. 5. Top fitness for current generation averaged over only 100 runs each, plotted for $\lambda = 500 * z, z \in \mathbb{N}, 0 \leq z < 20$ using genealogical diversity. The darker the color of the line, the higher is the depicted λ value.

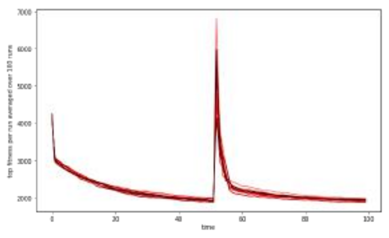


Fig. 6. Top fitness for current generation averaged over 100 runs each, plotted for $t = 2^z, z \in \mathbb{N}, 0 \leq z < 10$ using genealogical diversity. The darker the color of the line, the higher is the depicted t value.

Taken from original publication: Thomas Gabor, Lenz Belzner, Thomy Phan, and Kyrill Schmid. Preparing for the unexpected: Diversity improves planning resilience in evolutionary algorithms. In *15th IEEE International Conference on Autonomic Computing (ICAC)*, 2018

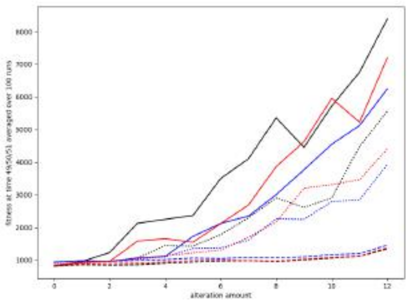


Fig. 7. Analysis of the fitness amplitude around unexpected change of varying intensity for the non-diverse (black), the genealogically diverse (red) and the domain-dependent diverse (blue) evolutionary algorithm respectively, plotted against the parameter A for the alternation amount of the change function c_A, all results averaged over 100 runs. The dashed line shows the population's top fitness value just before the change (generation 49). The solid line shows the top fitness just at the moment of unexpected change (generation 50). The dotted line shows the fitness one generation later (generation 51), when it has started to improve again.

non-diverse planner. Most remarkably, the domain-dependent variant manages to cope with changes $A \leq 4$ with almost no consequence for its performance.

VI. Related Work

The notion of diversity is researched in many different fields of science. In this Section, we discuss a few of these and their relation to the issue presented in this paper. To the author's knowledge, the issue of planning resilience is a rather novel one and applying genetic diversity to promote it is a unique contribution of this paper.

a) Planning and Reinforcement Learning: Rolling horizon genetic algorithms for online planning are widely used in real-time general video game playing [16], [17]. However, these approaches typically optimize with respect to the expectation of reward, thus they suffer from the drawbacks of non-diverse planning as discussed in this work. Statistical online planning has recently attracted a fair amount of research interest [23]–[25], also due to the successful combination with deep learning technology [26]–[28]. In general, diversity as a consideration for resilient planning is orthogonal to these approaches and could straightforwardly be combined with recent developments from the online planning literature.

We also see a relation to another recent line of research in reinforcement learning that explores ways of modeling aleatoric or epistemic uncertainty about future rewards or action selection mechanisms as distributions rather than by expectation [29]–[34]. This enables learning and decision making agents to explicitly deal with multimodal distributions of utility. It allows to incorporate risk and uncertainty into the decision making process, and to effectively deal with the exploration-exploitation tradeoff. In particular, distributional approaches foster the learning of diverse behavior, yielding robust transfer of learned skills to new, unseen situations [31].

b) Diversity in Software: In software engineering, diversity often takes the form of generating, offering, or using functionally equivalent variants of software artifacts during software development or deployment [35]. An extensive survey of current techniques is given in [19]. However, all of these differ from the approach in this paper in that we explicitly search for *functional* alternatives in the context of this paper, i.e., we want our diverse solutions to represent solutions to different problems (in order to possibly anticipate future problems) and not different solutions of equal quality to the same problem.

Still, the techniques presented in literature to exploit the prevalence of multiple instances of the same software artifact during runtime might be applied to variants generated by a diverse genetic algorithm as well. The work in this paper can be regarded as a first step to expose the population-based view of diversity within an automated search process with the process of software development. Similar trends in software engineering are discussed in [3], [36].

c) Diversity in Genetic Algorithms: Genetic algorithms make up a vast field of research. Regarding the basic definitions, this work follows the comprehensive description in [12]. Diversity has been recognized as an important indicator for good performance, although mainly applied to the static scenarios of offline adaptation: The authors of [37] provide an extensive survey of various methods to enforce diversity in genetic algorithms. These fall into the categories of external methods controlling the evolutionary process "from the outside" [18], [38] and methods integrating diversity as an additional objective into the genetic algorithm, using the concepts of multi-objective genetic optimization [39], [40]. Following the biological inspiration, the aptitude of genetic algorithms to an online setting with a changing environment has been thoroughly analyzed [41], [42].

The author of [14] describes a problem setting not unlike the one presented in this paper, i.e., the combination of maintaining diversity and searching in a changing environment. The issue of premature convergence is tackled by integrating a certain amount of random search into the genetic algorithm by performing hyper-mutation. This has since become standard procedure and is included in all genetic algorithms presented in this paper, which aims to further improve the resilience of the search process.

Most recently, the authors of [43] tackled the issue of using an evolutionary algorithm as an online planner for a complex software system. While they discuss high diversity as a key factor in achieving better re-planning results, they use diversity purely as an observation not as a direct goal of the evolutionary process. Instead, they too resort to an operator

Taken from original publication: Thomas Gabor, Lenz Belzner, Thomy Phan, and Kyrill Schmid. Preparing for the unexpected: Diversity improves planning resilience in evolutionary algorithms. In *15th IEEE International Conference on Autonomic Computing (ICAC)*, 2018

akin to high amounts of hyper-mutation to increase diversity by creating a new population that only inherits certain parts of the old population. To this end, the system must be able to directly recognize the event of an unexpected change after it has happened.

It is important to note that a lot of literature about diversity in genetic algorithms (or metaheuristic search in general) is concerned about covering the frontier of Pareto-optimal solutions in the search space [44]. The notion of diversity used in this paper, however, is a more genetic one and has as one of its main features that is *not* automatically derived from the nature of the fitness function. Interesting connections to game theory may still be made but are outside the scope of this work.

d) Resilience and Robustness: The preparation for unexpected or previously wrongly modeled change is an important issue for the practical application of machine learning in industry [4]. From an engineer's point of view, the diversity of the population of plans can be regarded as a typical *non-functional requirement (NFR)* with the cost of the plan representing the functional requirement. Applying NFR engineering processes to self-adaptive systems is still a new idea and a clear canon of relevant NFRs for these new challenges has not yet been found [2], [9].

VII. Conclusion

Since we expect future software systems to be increasingly self-adaptive and self-managing, we can also expect them to feature one or multiple components tasked with online planning. Online planning allows systems to learn to optimize their behavior in the face of a moving target fitness. However, it comes with a few pitfalls, one of which is the fact even small changes in the target fitness can have detrimental effects on the current plans' performance. It is thus imperative to keep an eye on a healthy level of diversity in our pool of alternative plans. As we have shown, this can severely soften the blow to overall performance, should only a few plans become impractical due to external circumstances.[7]

The diversity of a planner functions as a non-functional requirement for classic applications. Certain levels of desired diversity may be specified in order to augment system architectures that revolve around the optimization process of the system in order to provide flexibility on the component level [46]. This should be expected to strongly influence other properties commonly applied to complex self-adaptive systems like robustness or flexibility.

On an application level, the introduced concept of diversity-aware optimization may prove especially useful when the reduction in amplitude of fitness causes the system behavior to fall below a predefined quality threshold (or to do so more often at least). A diversity-aware planner might then be able to continue working as usual as its back-up plans fulfill the required quality agreement just as well while a non-diverse planner might more often feel the need to stop the execution of its plans (and thus halt the system in general) until it reaches a new plan of acceptable quality. In this case, we may formulate a non-functional requirement such as planning resilience, measuring how frequent and how big unexpected changes need to be in order to push the planner out of its quality requirements. Using the parameter λ, engineers can adjust the focus point of the planning component between performance and resilience optimization. How well statistical judgements can be made about said resilience property still needs to be evaluated, though.

It is up to future research to determine how the concept of diversity (especially genealogical diversity) generalizes for other optimization techniques like the cross-entropy method or simulated annealing. One way to integrate these techniques into the framework defined in this paper may be to set up a pool of solution candidates via ensemble learning [47].

Embracing diversity seems especially promising in search-based software testing (SBST) as test suites need to adapt faster to new possible exploits. In DevOps, developers push relatively small updates that need testing more frequently. Nonetheless, the changes applied to the code by the developer usually fall into the category of unexpected change as we defined it in this paper. That means, that diverse test generators could possible adapt quicker to the new software system under test. The mutual influence between diversity-aware evolutionary algorithms and co-evolutionary approaches[8] may be an interesting point of further research [21]. A likewise connection in biological systems has been found [48].

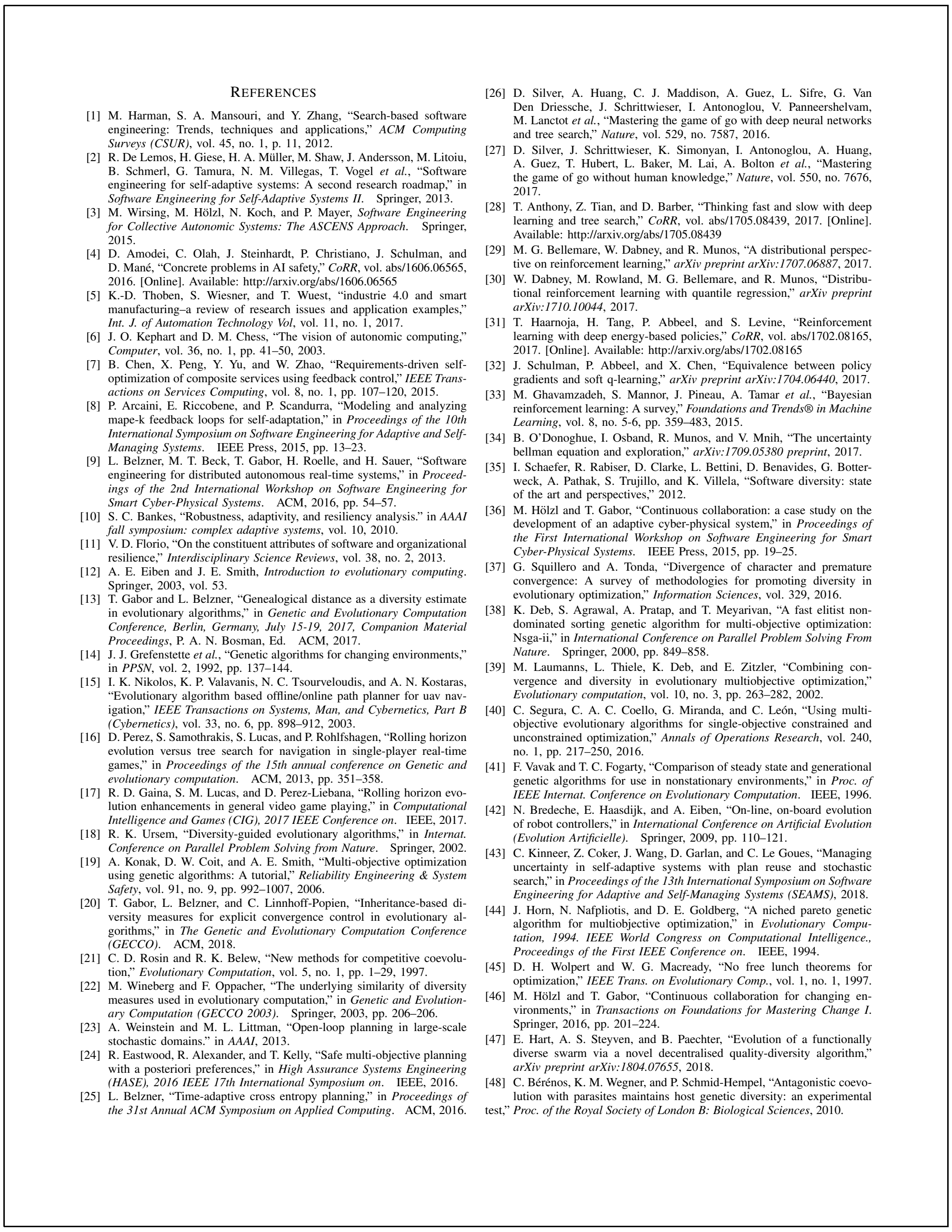
Many of the theoretical foundations explaining the ideal structure of a population for various optimization purposes are still unexplored. For instance, we assumed an unpredictable but neither explicitly hostile nor cooperative environment. Any scenario where the change occurs not only unexpected but intentional is likely to have fundamentally different properties.

We focused our study on the implications of using diversity within a planner and how the resilience to environmental change may be indicated in a quantifiable way. We have shown that diversity during planning can aid planning resilience in the face of change. Furthermore, we can employ such method in a domain-independent way using genealogical diversity and still achieve valuable results. Software engineering frameworks and processes are now needed to expose desired NFRs like planning resilience to the software and system design and test them adequately.

[7]It still holds that if we allow arbitrary changes in the environment, it is always possible to design a completely new fitness function so that any given instance of an evolutionary process becomes arbitrarily bad with respect to the new altered fitness function. This is due to the No-Free-Lunch theorem [45]. For realistic scenarios, however, there usually is a limit to how quickly and how drastically the fitness function is expected to change. A thorough analysis of those limits for some practical domains may present an interesting point for further research.

[8]For example, when SBST is used to analyze a system under test that is by itself capable of adapting and evolving, the complete testing cycle features two adversary evolutionary algorithms and is thus considered *co-evolutionary* [1].

Taken from original publication: Thomas Gabor, Lenz Belzner, Thomy Phan, and Kyrill Schmid. Preparing for the unexpected: Diversity improves planning resilience in evolutionary algorithms. In *15th IEEE International Conference on Autonomic Computing (ICAC)*, 2018



References

[1] M. Harman, S. A. Mansouri, and Y. Zhang, "Search-based software engineering: Trends, techniques and applications," *ACM Computing Surveys (CSUR)*, vol. 45, no. 1, p. 11, 2012.

[2] R. De Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, T. Vogel *et al.*, "Software engineering for self-adaptive systems: A second research roadmap," in *Software Engineering for Self-Adaptive Systems II*. Springer, 2013.

[3] M. Wirsing, M. Hölzl, N. Koch, and P. Mayer, *Software Engineering for Collective Autonomic Systems: The ASCENS Approach*. Springer, 2015.

[4] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, "Concrete problems in AI safety," *CoRR*, vol. abs/1606.06565, 2016. [Online]. Available: http://arxiv.org/abs/1606.06565

[5] K.-D. Thoben, S. Wiesner, and T. Wuest, "industrie 4.0 and smart manufacturing–a review of research issues and application examples," *Int. J. of Automation Technology Vol*, vol. 11, no. 1, 2017.

[6] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[7] B. Chen, X. Peng, Y. Yu, and W. Zhao, "Requirements-driven self-optimization of composite services using feedback control," *IEEE Transactions on Services Computing*, vol. 8, no. 1, pp. 107–120, 2015.

[8] P. Arcaini, E. Riccobene, and P. Scandurra, "Modeling and analyzing mape-k feedback loops for self-adaptation," in *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE Press, 2015, pp. 13–23.

[9] L. Belzner, M. T. Beck, T. Gabor, H. Roelle, and H. Sauer, "Software engineering for distributed autonomous real-time systems," in *Proceedings of the 2nd International Workshop on Software Engineering for Smart Cyber-Physical Systems*. ACM, 2016, pp. 54–57.

[10] S. C. Bankes, "Robustness, adaptivity, and resiliency analysis." in *AAAI fall symposium: complex adaptive systems*, vol. 10, 2010.

[11] V. D. Florio, "On the constituent attributes of software and organizational resilience," *Interdisciplinary Science Reviews*, vol. 38, no. 2, 2013.

[12] A. E. Eiben and J. E. Smith, *Introduction to evolutionary computing*. Springer, 2003, vol. 53.

[13] T. Gabor and L. Belzner, "Genealogical distance as a diversity estimate in evolutionary algorithms," in *Genetic and Evolutionary Computation Conference, Berlin, Germany, July 15-19, 2017, Companion Material Proceedings*, P. A. N. Bosman, Ed. ACM, 2017.

[14] J. J. Grefenstette *et al.*, "Genetic algorithms for changing environments," in *PPSN*, vol. 2, 1992, pp. 137–144.

[15] I. K. Nikolos, K. P. Valavanis, N. C. Tsourveloudis, and A. N. Kostaras, "Evolutionary algorithm based offline/online path planner for uav navigation," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 33, no. 6, pp. 898–912, 2003.

[16] D. Perez, S. Samothrakis, S. Lucas, and P. Rohlfshagen, "Rolling horizon evolution versus tree search for navigation in single-player real-time games," in *Proceedings of the 15th annual conference on Genetic and evolutionary computation*. ACM, 2013, pp. 351–358.

[17] R. D. Gaina, S. M. Lucas, and D. Perez-Liebana, "Rolling horizon evolution enhancements in general video game playing," in *Computational Intelligence and Games (CIG), 2017 IEEE Conference on*. IEEE, 2017.

[18] R. K. Ursem, "Diversity-guided evolutionary algorithms," in *Internat. Conference on Parallel Problem Solving from Nature*. Springer, 2002.

[19] A. Konak, D. W. Coit, and A. E. Smith, "Multi-objective optimization using genetic algorithms: A tutorial," *Reliability Engineering & System Safety*, vol. 91, no. 9, pp. 992–1007, 2006.

[20] T. Gabor, L. Belzner, and C. Linnhoff-Popien, "Inheritance-based diversity measures for explicit convergence control in evolutionary algorithms," in *The Genetic and Evolutionary Computation Conference (GECCO)*. ACM, 2018.

[21] C. D. Rosin and R. K. Belew, "New methods for competitive coevolution," *Evolutionary Computation*, vol. 5, no. 1, pp. 1–29, 1997.

[22] M. Wineberg and F. Oppacher, "The underlying similarity of diversity measures used in evolutionary computation," in *Genetic and Evolutionary Computation (GECCO 2003)*. Springer, 2003, pp. 206–206.

[23] A. Weinstein and M. L. Littman, "Open-loop planning in large-scale stochastic domains." in *AAAI*, 2013.

[24] R. Eastwood, R. Alexander, and T. Kelly, "Safe multi-objective planning with a posteriori preferences," in *High Assurance Systems Engineering (HASE), 2016 IEEE 17th International Symposium on*. IEEE, 2016.

[25] L. Belzner, "Time-adaptive cross entropy planning," in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. ACM, 2016.

[26] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, 2016.

[27] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, 2017.

[28] T. Anthony, Z. Tian, and D. Barber, "Thinking fast and slow with deep learning and tree search," *CoRR*, vol. abs/1705.08439, 2017. [Online]. Available: http://arxiv.org/abs/1705.08439

[29] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," *arXiv preprint arXiv:1707.06887*, 2017.

[30] W. Dabney, M. Rowland, M. G. Bellemare, and R. Munos, "Distributional reinforcement learning with quantile regression," *arXiv preprint arXiv:1710.10044*, 2017.

[31] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, "Reinforcement learning with deep energy-based policies," *CoRR*, vol. abs/1702.08165, 2017. [Online]. Available: http://arxiv.org/abs/1702.08165

[32] J. Schulman, P. Abbeel, and X. Chen, "Equivalence between policy gradients and soft q-learning," *arXiv preprint arXiv:1704.06440*, 2017.

[33] M. Ghavamzadeh, S. Mannor, J. Pineau, A. Tamar *et al.*, "Bayesian reinforcement learning: A survey," *Foundations and Trends® in Machine Learning*, vol. 8, no. 5-6, pp. 359–483, 2015.

[34] B. O'Donoghue, I. Osband, R. Munos, and V. Mnih, "The uncertainty bellman equation and exploration," *arXiv:1709.05380 preprint*, 2017.

[35] I. Schaefer, R. Rabiser, D. Clarke, L. Bettini, D. Benavides, G. Botterweck, A. Pathak, S. Trujillo, and K. Villela, "Software diversity: state of the art and perspectives," 2012.

[36] M. Hölzl and T. Gabor, "Continuous collaboration: a case study on the development of an adaptive cyber-physical system," in *Proceedings of the First International Workshop on Software Engineering for Smart Cyber-Physical Systems*. IEEE Press, 2015, pp. 19–25.

[37] G. Squillero and A. Tonda, "Divergence of character and premature convergence: A survey of methodologies for promoting diversity in evolutionary optimization," *Information Sciences*, vol. 329, 2016.

[38] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii," in *International Conference on Parallel Problem Solving From Nature*. Springer, 2000, pp. 849–858.

[39] M. Laumanns, L. Thiele, K. Deb, and E. Zitzler, "Combining convergence and diversity in evolutionary multiobjective optimization," *Evolutionary computation*, vol. 10, no. 3, pp. 263–282, 2002.

[40] C. Segura, C. A. C. Coello, G. Miranda, and C. León, "Using multi-objective evolutionary algorithms for single-objective constrained and unconstrained optimization," *Annals of Operations Research*, vol. 240, no. 1, pp. 217–250, 2016.

[41] F. Vavak and T. C. Fogarty, "Comparison of steady state and generational genetic algorithms for use in nonstationary environments," in *Proc. of IEEE Internat. Conference on Evolutionary Computation*. IEEE, 1996.

[42] N. Bredeche, E. Haasdijk, and A. Eiben, "On-line, on-board evolution of robot controllers," in *International Conference on Artificial Evolution (Evolution Artificielle)*. Springer, 2009, pp. 110–121.

[43] C. Kinneer, Z. Coker, J. Wang, D. Garlan, and C. Le Goues, "Managing uncertainty in self-adaptive systems with plan reuse and stochastic search," in *Proceedings of the 13th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 2018.

[44] J. Horn, N. Nafpliotis, and D. E. Goldberg, "A niched pareto genetic algorithm for multiobjective optimization," in *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*. IEEE, 1994.

[45] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Trans. on Evolutionary Comp.*, vol. 1, no. 1, 1997.

[46] M. Hölzl and T. Gabor, "Continuous collaboration for changing environments," in *Transactions on Foundations for Mastering Change I*. Springer, 2016, pp. 201–224.

[47] E. Hart, A. S. Steyven, and B. Paechter, "Evolution of a functionally diverse swarm via a novel decentralised quality-diversity algorithm," *arXiv preprint arXiv:1804.07655*, 2018.

[48] C. Bérénos, K. M. Wegner, and P. Schmid-Hempel, "Antagonistic coevolution with parasites maintains host genetic diversity: an experimental test," *Proc. of the Royal Society of London B: Biological Sciences*, 2010.

Taken from original publication: Thomas Gabor, Thomy Phan, and Claudia Linnhoff-Popien. Productive fitness in diversity-aware evolutionary algorithms. *Natural Computing*, pages 1–14, 2021

Natural Computing (2021) 20:363–376
https://doi.org/10.1007/s11047-021-09853-3

Productive fitness in diversity-aware evolutionary algorithms

Thomas Gabor[1] · Thomy Phan[1] · Claudia Linnhoff-Popien[1]

Accepted: 15 March 2021 / Published online: 29 April 2021

Abstract
In evolutionary algorithms, the notion of diversity has been adopted from biology and is used to describe the distribution of a population of solution candidates. While it has been known that maintaining a reasonable amount of diversity often benefits the overall result of the evolutionary optimization process by adjusting the exploration/exploitation trade-off, little has been known about what diversity is optimal. We introduce the notion of productive fitness based on the effect that a specific solution candidate has some generations down the evolutionary path. We derive the notion of final productive fitness, which is the ideal target fitness for any evolutionary process. Although it is inefficient to compute, we show empirically that it allows for an *a posteriori* analysis of how well a given evolutionary optimization process hit the ideal exploration/exploitation trade-off, providing insight into *why* diversity-aware evolutionary optimization often performs better.

Keywords Evolutionary algorithm · Adaptive fitness · Diversity

1 Introduction

Evolutionary algorithms are a widely used type of stochastic optimization that mimics biological evolution in nature. Like any other metaheuristic optimization algorithm (Brown et al. 2005; Conti et al. 2018), they need to maintain a balance on the exploration/exploitation trade-off in their search process: High exploration bears the risk to miss out on optimizing the intermediate solutions to the fullest; high exploitation bears the risk to miss the global optimum and get stuck in a sub-optimal part of the search space. Analogous to biological evolution, diversity within the population of solution candidates has been identified as a central feature to adjust the exploration/exploitation trade-off. Many means to maintain the diversity of the population throughout the process of evolution have been developed in literature; comprehensive overviews are provided by Squillero and Tonda (2016) and Gabor et al. (2018), for example.

For problems with complex fitness landscapes, it is well known that increased exploration (via increased diversity) yields better overall results in the optimization, even when disregarding any diversity goal in the final evaluation (Ursem 2002; Toffolo and Benini 2003). However, this gives rise to a curious phenomenon: By augmenting the fitness function and thus making it match the original objective function *less*, we actually get results that optimize the original objective function *more*. This implies that any evolutionary algorithm does not immediately optimize for the fitness function it uses (but instead optimizes for a slightly different implicit goal). Furthermore, to really optimize for a given objective function, one should ideally use a (slightly) different fitness function for evolution. In this paper, we introduce *final productive fitness* as a theoretical approach to derive the ideal fitness function from a given objective function.

We see that final productive fitness cannot feasibly be computed in advance. However, we show how to approximate it a posteriori, i.e., when the optimization process is already finished. We show that the notion of final productive fitness is sound by applying it to the special case of diversity-aware evolutionary algorithms, which (for our

✉ Thomas Gabor
thomas.gabor@ifi.lmu.de

Thomy Phan
thomy.phan@ifi.lmu.de

Claudia Linnhoff-Popien
linnhoff@ifi.lmu.de

[1] LMU Munich, Oettingenstraße 67, 80538 München, Germany

Taken from original publication: Thomas Gabor, Thomy Phan, and Claudia Linnhoff-Popien. Productive fitness in diversity-aware evolutionary algorithms. *Natural Computing*, pages 1–14, 2021

purposes) are algorithms that directly encode a strife for increased diversity by altering the fitness of the individuals. By running these on various benchmark problems, we empirically show that diversity-aware evolutionary processes might just approximate final productive fitness more accurately than an evolutionary process using just the original objective. We show that the fitness alteration performed by these algorithms, when it improves overall performance, does so *while* (perhaps *because*) it better approximates final productive fitness. We thus argue that the notion of final productive fitness for the first time provides a model of *how* diversity is beneficial to evolutionary optimization, which has been called for by various works in literature:

- "One of the urgent steps for future research work is to better understand the influence of diversity for achieving good balance between exploration and exploitation." (Črepinšek et al. 2013),
- "This tendency to discover both quality and diversity at the same time differs from many of the conventional algorithms of machine learning, and also thereby suggests a different foundation for inferring the approach of greatest potential for evolutionary algorithms." (Pugh et al. 2016),
- "However, the fragmentation of the field and the difference in terminology led to a general dispersion of this important corpus of knowledge in many small, hard-to-track research lines" and, "[w]hile diversity preservation is essential, the main challenge for scholars is devising general methodologies that could be applied seamlessly [...]" (Squillero and Tonda 2016).

It should be noted that the approach presented in this paper merely provides a new perspective on exploration/exploitation in evolutionary algorithms and a new method of analyzing the effects of diversity. It is up to future works to derive new means to actively promote diversity from this analysis.

In this paper, we provide a short mathematical description of evolutionary processes in Sect. 2 and build our notion of (final) productive fitness on top of that in Sect. 3. Section 4 describes the empirical results and Sect. 5 discusses related work before Sect. 6 concludes.

2 Foundations

For this paper, we assume an evolutionary process (EP) to be defined as follows: Given a fitness function $f : \mathcal{X} \to [0;1] \subset \mathbb{R}$ for an arbitrary set $\mathcal{X}$ called the search space, we want to find an individual $x \in \mathcal{X}$ with the best fitness *f*(*x*). For a maximization problem, the best fitness is that of an individual x so that $f(x) \geq f(x')\ \forall x' \in \mathcal{X}$. For a minimization problem, the best fitness is that of an individual x so that $f(x) \leq f(x')\ \forall x' \in \mathcal{X}$. Note that we normalize our fitness space on $[0;1] \subset \mathbb{R}$ for all problems for ease of comparison. Whenever the maximum and minimum fitness are bounded, this can be done without loss of generality.

Usually, the search space $\mathcal{X}$ is too large or too complicated to guarantee that we can find the exact best individual(s) using standard computing models (and physically realistic time). Thus, we take discrete subsets of the search space $\mathcal{X}$ via sampling and iteratively improve their fitness. An evolutionary process $\mathcal{E}$ over g generations, $g \in \mathbb{N}$, is defined as $\mathcal{E} = \langle \mathcal{X}, e, f, (X_i)_{i<g} \rangle$. $\mathcal{X}$ is the search space. $e : 2^{\mathcal{X}} \to 2^{\mathcal{X}}$ is the evolutionary step function so that $X_{i+1} = e(X_i)\ \forall i \geq 0$. As defined above, $f : \mathcal{X} \to [0;1] \subset \mathbb{R}$ is the fitness function. $(X_i)_{i<g}$ is a series of populations so that $X_i \subseteq \mathcal{X}\ \forall i$ and X_0 is the initial population. Note that as the evolutionary step function e is usually non-deterministic, we define $E(X) = \{X' | X' = e(X)\}$ to be the set of all possible next populations.

We use the following evolutionary operators:

- The recombination operator $rec : \mathcal{X} \times \mathcal{X} \to \mathcal{X}$ generates a new individual from two individuals.
- The mutation operator $mut : \mathcal{X} \to \mathcal{X}$ alters a given individual slightly to return a new one.
- The migration operator $mig : \mathcal{X}$ generates a random individual $mig() \in \mathcal{X}$.
- The (survivors) selection operator $sel : 2^{\mathcal{X}} \times \mathbb{N} \to 2^{\mathcal{X}}$ returns a new population $X' = sel(X, n)$ given a population $X \subseteq \mathcal{X}$, so that $|X'| \leq n$.

The operators rec, mut, mig can be applied to a population X by choosing individuals from X to fill their parameters (if any) according to some selection scheme $\sigma : 2^{\mathcal{X}} \to 2^{\mathcal{X}}$ and adding their return to the population. For example, we allow to write $mut_\sigma(X) = X \cup \{\, mut(x') \mid x' \in \sigma(X)\,\}$. Note that all children are added to the population and do not replace their parents in this formulation.

For any evolutionary process $\mathcal{E} = \langle \mathcal{X}, e, f, (X_i)_{i<g} \rangle$ and selection schemes $\sigma_1, \sigma_2, \sigma_3$ we assume that

$$X_{i+1} = e(X_i) = sel\,(mig_{\sigma_3}(mut_{\sigma_2}(rec_{\sigma_1}(X_i))), |X_i|). \tag{1}$$

Usually, we assume that an evolutionary process fulfills its purpose if the best fitness of the population tends to become better over time, i.e., given a sufficiently large amount of generations $k \in \mathbb{N}$, it holds for maximization problems that $\max_{x \in X_i} f(x) < \max_{x \in X_{i+k}} f(x)$. We define the overall result of an evolutionary process $\mathcal{E} = \langle \mathcal{X}, e, f, (X_i)_{i<g} \rangle$ with respect to a fitness function ϕ (which may or may not be different from the fitness f used during evolution) to be best value found and kept in evolution, i.e., for a maximizing objective ϕ we define

Taken from original publication: Thomas Gabor, Thomy Phan, and Claudia Linnhoff-Popien. Productive fitness in diversity-aware evolutionary algorithms. *Natural Computing*, pages 1–14, 2021

$$|\mathcal{E}|_\phi = \max_{x \in X_g} \phi(x). \tag{2}$$

Note that there are evolutionary processes which include a *hall-of-fame* mechanism, i.e., are able to return the result fitness

$$||\mathcal{E}||_\phi = \max_{i=1,\ldots,g} \; \max_{x \in X_i} \; \phi(x). \tag{3}$$

However, we can derive the equality $|\mathcal{E}|_\phi = ||\mathcal{E}||_\phi$ when we assume *elitism* with respect to ϕ, i.e., $\arg\max_{x \in X_i} \phi(x) \in X_{i+1}$ for all $i = 1, \ldots, g$. Since it makes reasoning easier and hardly comes with any drawback for sufficiently large populations, we use elitist evolutionary processes (with respect to f) from here on.

3 Approach

The central observation we build our analysis on is that in many cases the results of optimizing for a given objective function (called OF) can be improved by not using OF as a the fitness function f of the evolutionary process directly. Consequently, changing the fitness function f away from the true objective OF in some cases leads to better results with respect to the original objective function OF. Note that this phenomenon extends beyond just heuristic optimization and is known as *reward shaping* in reinforcement learning, for example (Ng et al. 1999).

In evolutionary algorithms oftentimes a property called diversity is considered in addition to the objective function OF to improve the progress of the evolutionary process (Gabor et al. 2018; Squillero and Tonda 2016; Ursem 2002). In some way or the other, diversity-enhancing evolutionary algorithms award individuals of the population for being different from other individuals in the population. While there are many ways to implement this behavior, like topology-based methods (Tomassini 2006), fitness sharing (Sareni and Krahenbuhl 1998), ensembling (Hart and Sim 2018), etc., we consider an instance of diversity-enhancing evolutionary algorithms that is simpler to analyze: By quantifying the distance of a single individual to the population, we can define a secondary fitness SF that rewards high diversity in the individual. This approach was shown by Wineberg and Oppacher (2003) to be an adequate general representation of most well-known means of measuring diversity in a population.

In order to avoid the difficulties of multi-objective evolution, we can then define the augmented fitness function AF that incorporates both the objective fitness OF and the secondary fitness SF into one fitness function to be used for the evolutionary process.

Definition 1 (Augmented Fitness) Given the objective fitness OF, a diversity-aware secondary fitness SF, and a diversity weight $\lambda \in [0;1] \subset \mathbb{R}$, we define the augmented fitness AF as

$$\mathrm{AF}(x) = (1-\lambda) \cdot \mathrm{OF}(x) + \lambda \cdot \mathrm{SF}(x). \tag{4}$$

As is shown in Gabor et al. (2018) and Wineberg and Oppacher (2003) such a definition of the augmented fitness suffices to show benefits of employing diversity.

We can then define two evolutionary processes $\mathcal{E}_{\mathrm{OF}} = \langle \mathcal{X}, e, \mathrm{OF}, (X_i)_{i<g} \rangle$ and $\mathcal{E}_{\mathrm{AF}} = \langle \mathcal{X}, e, \mathrm{AF}, (X'_i)_{i<g} \rangle$. We observe the curious phenomenon that in many cases the augmented fitness AF better optimizes for OF than using OF itself, formally

$$|\mathcal{E}_{\mathrm{OF}}|_{\mathrm{OF}} < |\mathcal{E}_{\mathrm{AF}}|_{\mathrm{OF}}, \tag{5}$$

which raises the following question: If OF is not the ideal fitness function to optimize for the objective OF, what is?

Given a sequence of populations $(X_i)_{i<g}$ spanning over multiple generations $i = 1, \ldots, g$ we can write down what we actually want our population to be like inductively starting from the last generation g: The net benefit of X_g to our (maximizing) optimization process is exactly

$$|\mathcal{E}|_{\mathrm{OF}} = \max_{x \in X_g} \mathrm{OF}(x) \tag{6}$$

as this population will not evolve any further and thus the best individual within X_g is what we are going to be stuck with as the result of the optimization process.

Note that the individuals of X_{g-1} already contribute differently to the result of the optimization process: From the perspective of generation $g-1$ the overall optimization result is

$$\max_{x \in X_{g-1}} \; \max_{x' \in X_g(x)} \; \mathrm{OF}(x') \tag{7}$$

where the follow-up generation $X_g(x)$ is any[1] population from $\{X_g \mid X_g \in E(X_{g-1}) \wedge x \in X_g\}$, i.e., the possible next populations where x survived.

Intuitively, the contribution of the the second-to-last generation X_{g-1} to the result of the optimization process stems from the objective fitness OF that this generation's individuals can still achieve in the final generation X_g. Generally, this does not fully coincide with the application of the objective function OF in said generation:

$$\max_{x \in X_{g-1}} \; \max_{x' \in X_g(x)} \; \mathrm{OF}(x') \;\; \neq \;\; \max_{x \in X_{g-1}} \mathrm{OF}(x) \tag{8}$$

[1] Arguments can be made to pick either the average or the maximum over all these populations. We discuss both cases later in the text.

Taken from original publication: Thomas Gabor, Thomy Phan, and Claudia Linnhoff-Popien. Productive fitness in diversity-aware evolutionary algorithms. *Natural Computing*, pages 1–14, 2021

That means: While rating individuals according to their objective fitness OF in the last generation of the evolutionary process is adequate, the actual benefit of the individual x to the optimization result and the value of $\text{OF}(x)$ may diverge more the earlier we are in the evolutionary process. Accordingly, at the beginning of an evolutionary process, the objective fitness OF might not be a good estimate of how much the individuals will contribute to the process's return with respect to OF at the end of the optimization process. Still, standard optimization techniques often use the objective fitness OF as a (sole) guideline for the optimization process.

Instead, we ideally want to make every decision (mutation, recombination, survival, ...) at every generation X_i with the ideal result for the following generations $X_{i+1}, X_{i+2}, \dots$ and ultimately the final generation X_g in mind. We call this the *optimal evolutionary process*. Obviously, to make the optimal decision early on, we would need to simulate all the way to the end of the evolution, including all the follow-up decisions. This renders optimal evolution infeasible as an algorithm. However, we can use it for a posteriori analysis of what has happened within a different evolutionary process. In order to do so, we need to give a fitness function for the optimal process (as it obviously should not be OF).

Instead, we formalize the benefit to the optimization process discussed above and thus introduce the notion of productive fitness. But first, we need a simple definition on the inter-generational relationships between individuals.

Definition 2 (Descendants) Given an individual x in the population of generation i, $x \in X_i$, of an evolutionary process. All individuals $x' \in X_{i+1}$ so that x' resulted from x via a mutation operator, i.e., $x' = \text{mut}(x)$, or a recombination operator with any other parent, i.e., there exists $y \in X_i$ so that $x' = \text{rec}(x, y)$, are called *direct descendants* of x. Further given a series of populations $(X_i)_{0<i<g}$ we define the set of all descendants D_x as the transitive hull on all direct descendants of x.

We can now use this relationship to assign the benefit that a single individual has had to the evolution a posteriori. For this, we simply average the fitness of all its surviving descendants.

Definition 3 (Productive Fitness) Given an individual x in the population of generation i, $x \in X_i$, of an evolutionary process. Let $D_x \subseteq \mathcal{X}$ be the set of all descendants from x. The *productive fitness after n generations* or *n-productive fitness* is the average objective fitness of x's descendants, written

$$\text{PF}_n(x) = \begin{cases} \text{avg}_{x' \in D_x \cap X_{i+n}} \text{OF}(x') & \textit{if } D_x \cap X_{i+n} \neq \emptyset \\ w & \textit{otherwise.} \end{cases} \tag{9}$$

Note that in case the individual x has no descendants in n generations, we set its productive fitness $\text{PF}_n(x)$ to a worst case value w, which in our case of bounded fitness values is 0 for maximizing optimization processes and 1 for minimizing optimization processes.

We argue that the productive fitness PF is better able to describe the actual benefit the individual brings to the optimization process, as represented by what parts of the individual still remain inside the population in a few generations. Note that our notion of productive fitness is rather harsh in two points:

- We only take the average of all descendants' fitness. One could argue that we may want a more optimistic approach where we might reward the individual for the best offspring it has given rise to. However, we argue that every bad individual binds additional resources for eliminating it down the road and thus a low target accuracy should actively be discouraged.
- When the line of an individual dies out completely, we assign the worst possible fitness. Arguments could be made that even dead lines contribute to the search process by ruling out unpromising areas while, e.g., increasing the diversity scores of individuals in more promising areas of the search space. Still, we do count any however distant descendants, so even small contributions to the final population avoid the penalty w.

We leave the analysis of the effects of the discussed parameters to future work. Note that for now, our notion of productive fitness only covers a fixed horizon into the future. We can trivially extend this definition to respect the final generation no matter what generation the current individual is from:

Definition 4 (Final Productive Fitness) Given an individual x in the population of generation i, $x \in X_i$, of an evolutionary process of g generations in total. The final productive fitness of x is the fitness of its descendants in the final generation, i.e., $\text{FPF}(x) = \text{PF}_{g-i}(x)$.

We argue that final productive fitness is able to describe what the fitness function of an optimal evolutionary process looks like: Every evaluation is done in regard to the contribution to the final generation, i.e., the ultimate solution returned by the search process.

Thesis 1 When rolling the ideal choices in all randomized evolutionary operators, final productive fitness FPF is the optimal fitness function for evolutionary processes, i.e., an evolutionary process yields the best results when it optimizes for FPF at every generation.

We sketch a short argument in favor of Thesis 1. For a more in-depth discussion, see Gabor and Linnhoff-Popien

Taken from original publication: Thomas Gabor, Thomy Phan, and Claudia Linnhoff-Popien. Productive fitness in diversity-aware evolutionary algorithms. *Natural Computing*, pages 1–14, 2021

(2020). Let $\mathcal{E}_{\text{FPF}} = \langle \mathcal{X}, e, \text{FPF}, (X_i^{\text{FPF}})_{i<g} \rangle$ be an evolutionary process using final productive fitness FPF. Let $\mathcal{E}_{\text{IDF}} = \langle \mathcal{X}, e, \text{IDF}, (X_i^{\text{IDF}})_{i<g} \rangle$ be an evolutionary process using a different (possibly more ideal) fitness IDF. Let $X_0^{\text{FPF}} = X_0^{\text{IDF}}$. We assume that

$$\max_{x \in X_g^{\text{FPF}}} \text{OF}(x) < \max_{x \in X_g^{\text{IDF}}} \text{OF}(x). \tag{10}$$

Since Eq. 10 implies that at least $X_g^{\text{FPF}} \neq X_g^{\text{IDF}}$, there is an individual $x \in X_g^{\text{IDF}}$ so that $x \notin X_g^{\text{FPF}}$ and $\text{OF}(x) > \max_{y \in X_g^{\text{FPF}}} \text{OF}(y)$. Since both $\mathcal{E}_{\text{FPF}}$ and $\mathcal{E}_{\text{IDF}}$ use the same evolutionary step function e except for the used fitness, their difference regarding x needs to stem from the fact that there exists an individual x' that is an ancestor of x, i.e., $x \in D_{x'}$, so that x' was selected for survival in $\mathcal{E}_{\text{IDF}}$ and not in $\mathcal{E}_{\text{FPF}}$, which implies that $\text{FPF}(x') < \text{IDF}(x')$. However, since x is a possible descendant for x', the computation of $\text{FPF}(x')$ should have taken $\text{OF}(x)$ into account,[2] meaning that x' should have survived in $\mathcal{E}_{\text{FPF}}$ after all, which contradicts the previous assumption. □

Of course, Thesis 1 is a purely theoretical argument as we cannot guarantee optimal choices in usually randomized evolutionary operators and productive fitness in general thus comes with the reasonable disadvantage that it cannot be fully computed in advance. But for a given, completed run of an evolutionary process, we can compute the factual FPF single individuals had a posteriori. There, we still do not make optimal random choices but just assume the ones made as given.

Still, we take Thesis 1 as hint that final productive fitness might be the right target to strive for. We argue that augmenting the objective fitness OF (even with easily computable secondary fitness functions) may result in a fitness function which better approximates final productive fitness FPF. In the following Sect. 4, we show empirically that (in the instances where it helps[3]) diversity-based secondary fitness SF resembles the final productive fitness FPF of individuals much better than the raw objective function OF does.

Thesis 2 When a diversity-aware augmented fitness function AF is aiding the evolutionary optimization process with respect to an objective fitness OF, it is doing so by approximating the final productive fitness FPF of a converged evolutionary process in a more stable way (i.e., more closely when disregarding the respective scaling of the fitness functions) throughout the generations of the evolutionary process.

This connection not only explains why diversity-aware fitness functions fare better than the pure objective fitness but also poses a first step towards a description how to deliberately construct diversity-aware fitness functions, knowing that their ideal purpose is to approximate the not fully computable final productive fitness. Again, we refer to Gabor and Linnhoff-Popien (2020) for more elaborate theoretical arguments.

Since we cannot estimate all possible futures for an evolutionary process, we provide empirical evidence in favor of Thesis 2 using a a posteriori approximation: Given an already finished evolutionary process, we compute the FPF values given only those individuals that actually came into being during that single evolutionary process (instead of using all possible descendants). We argue that this approximation is valid because *if* the evolutionary process was somewhat successful, then all individuals' descendants should be somewhat close to their ideal descendants *most likely*.[4] Note that the reverse property is not true (i.e., even in a bad run, individuals still aim to generate better descendants, not worse), which is why our approximation does not permit any statements about augmented fitness that does not aid the evolutionary process.

4 Experiments

For all experiments, we run an evolutionary process as defined in Sect. 2 with a mutation operator *mut* that adds a (possibly negative) random value to one dimension of individual, applied with rate 0.1 to all individuals at random. For *rec* we apply random crossover with rate 0.3 for a single individual and a randomly chosen mate. We apply *mig* with a rate of 0.1 (Gabor et al. 2018). Following Wineberg and Oppacher (2003) and the results in Gabor et al. (2018), we focus on a Manhattan distance function for the secondary fitness; we also plot evolutionary processes using fitness sharing with parameter $\alpha = 2.0$ and dissimilarity threshold $\sigma = n$, where n is the dimensionality of the problem (Sareni and Krahenbuhl 1998), or inherited fitness with inheritance weight $\kappa = 0.5$ (Chen et al. 2002; Gabor et al. 2018) for comparison,

[2] Note that $\text{OF}(x)$ cannot be compensated by other descendants of x' with possibly bad objective fitness even as we average the results because all offspring is created by a random choice, which we assume to be ideal. This also shows how strong that assumption is.

[3] Note the gravity of that restriction: We do not consider failed runs of evolutionary algorithms since we have no assumptions on how FPF *should* behave. i.e., relate to AF, in that case. Future work may fill that void.

[4] Note that we could construct a terrible evolutionary process that just happens to find the global optimum in the last generation out of the blue via random migration. That process would have a poor stability between AF and FPF but a very successful result. However, since evolution at every step tries not to be terrible, we consider that scenario to be quite unlikely so that it should not play a role when we analyze the augmented fitness on multiple runs, parametrizations, and domains.

Taken from original publication: Thomas Gabor, Thomy Phan, and Claudia Linnhoff-Popien. Productive fitness in diversity-aware evolutionary algorithms. *Natural Computing*, pages 1–14, 2021

since both approaches also use an adapted fitness function to promote diversity.[5] The selection operator *sel* is a simple rank-based cut-off in the shown evolutionary processes. Cut-off with protection for new individuals as well as roulette wheel selection was also tested without yielding noticeably different results.

All code that produced the results of this paper is available at github.com/thomasgabor/naco-evolib.

4.1 Pathfinding

We start with the pathfinding problem, which was shown to greatly benefit from employing diversity in the optimization process (Gabor et al. 2018): Given a room of dimensions 1×1, we imagine a robot standing at position (0.5, 0.1). It needs to reach a target area at the opposite side of the room. See Fig. 1 for an illustration. The room also features a huge obstacle in the middle and thus the robot needs to decide on a way around it. The agent can move by performing an action $a \in \{(\delta x, \delta y) | -0.33 < \delta x < 0.33, -0.33 < \delta y < 0.33\}$. A single solution candidate consists of $n = 5$ actions $\langle(\delta x_i, \delta y_i)\rangle_{1 \leq i \leq n}$. It achieves a reward of $\frac{1}{n} = 0.2$ every time it stays within the target area between steps, i.e., its fitness is given via $\textsc{of}(\langle(\delta x_i, \delta y_i)\rangle_{1 \leq i \leq m}) = r(\langle(\delta x_i, \delta y_i)\rangle_{1 \leq i \leq m}, (0.5, 0.1))$ where

$$r(\langle(\delta x_i, \delta y_i)\rangle_{1 \leq i \leq m}, (x, y)) = r(\langle(\delta x_i, \delta y_i)\rangle_{2 \leq i \leq m}, (x + \delta x_1, y + \delta y_1)) + t(x, y) \tag{11}$$

$$\text{with } r(\langle\rangle, (x, y)) = t(x, y) \tag{12}$$

$$\text{and } t(x, y) = \begin{cases} \frac{1}{n} & if 0.4 \leq x \leq 0.6 \land 0.8 \leq y \leq 1.0 \\ 0 & otherwise. \end{cases} \tag{13}$$

The pathfinding problem lends itself to the application of diversity, as the optimization process in most cases first strikes a local optimum where it reaches the target area sometime by accident (and most probably towards the end of its steps). It then needs to switch to the global optimum where the first three steps are as goal-directed as possible and the last two steps are very small in order to stay within the target area.

We now compare a standard evolutionary algorithm given only the objective function $\textsc{of}(x) = f(x)$ to a diversity-aware evolutionary algorithm using the Manhattan distance on the solution candidate structure as a secondary

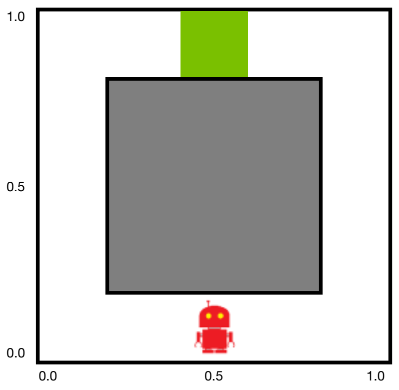


Fig. 1 The Pathfinding problem. A robot (red) needs to find a fixed path to reach the target area (green)

fitness function, i.e., $\textsc{af}(x) = (1 - \lambda) \cdot \textsc{of}(x) + \lambda \cdot \textsc{sf}(x)$ as given in Definition 1

$$\text{where } \textsc{sf}(x) = \frac{1}{2n} \cdot \operatorname{avg}_{x' \in \sigma_4(X)} manhattan(x, x') \tag{14}$$

$$\text{and } manhattan(\langle(\delta x_i, \delta y_i)\rangle_{1 \leq i \leq m}, \langle(\delta x'_i, \delta y'_i)\rangle_{1 \leq i \leq m}) = \sum_{i=1}^{m} |\delta x_i - \delta x'_i| + |\delta y_i - \delta y'_i|. \tag{15}$$

Note that σ_4 is a selection function that randomly selects 10 individuals from the population X. We use it to reduce the computational cost of computing the pairwise distance for all individuals in the population. Its admissibility for approximating the full pairwise distance was shown in Gabor and Belzner (2017). Just as we normalized the fitness function f to $[0; 1] \subset \mathbb{R}$ we also normalize the secondary fitness $\textsc{sf}$ to the same range via division by the maximum Manhattan distance between two individuals, i.e., $2n$, to make the combination easier to understand. For now, we set $\lambda = 0.4$, which we discuss later.

Each evolution was run 30 times for 1500 generations each, using a population size of 50. Figure 2a shows the best fitness achieved per generation for all tested approaches. We see that (especially distance-based) diversity-aware evolution produces much better objective results. Figure 2b shows the separate diversity score $\textsc{sf}$ maintained by the best individual, which can only be computed in a meaningful way for Manhattan diversity. In Fig. 2c the standard approach shows the same plot as before since its fitness is not augmented. For all other approaches we plot

[5] Of course, many more diversity-aware evolutionary algorithms could have been analyzed here. We would like to refer to our experiments in Gabor et al. (2018) as well as the survey by Squillero and Tonda (2016) for a comprehensive overview.

Taken from original publication: Thomas Gabor, Thomy Phan, and Claudia Linnhoff-Popien. Productive fitness in diversity-aware evolutionary algorithms. *Natural Computing*, pages 1–14, 2021

the augmented fitness AF that is actually used for selection. We see that due to the combination of distance and objective fitness, Manhattan-diverse evolution starts higher but climbs slower than the respective objective fitness. Fitness sharing results in very small absolute values for fitness but climbs up nonetheless.

From the already run evolutionary processes, we can compute the final productive fitness as given in Definitions 3 and 4 a posteriori. Figure 2d shows the maximum FPF per generation. We see that Manhattan-diverse and inherited fitness maintain a rather continuous lineage from the initial population to the best solution in the final generation as the final fitness propagates to the final productive fitness of very early generations. This behavior is rather unsurprising but illustrates the notion of the final productive fitness that measures the individuals' impact in the final result.

For Fig. 2e we compute the perhaps most interesting measurement: This plot shows for each population X in a given generation the result of $\text{avg}_{x \in X} |\text{AF}(x) - \text{FPF}(x)|$, i.e., the average difference between the augmented fitness and the final productive fitness *per individual*. Thus, we get to assess how well the augmented fitness approximates the final productive fitness. There are a few observations to be made:

1. Towards the last few generations, we notice a rapid spike in the FPF as the amount of descendants in the final generation to be considered for the FPF decreases fast.
2. The actual value of the distance (i.e., the height of the line) is irrelevant to the analysis of Thesis 2 and largely determined by the setting of λ.
3. Throughout most of the plot, the Manhattan-diverse evolution maintains a relatively stable level, i.e., the augmented fitness AF approximates the final productive fitness FPF throughout the evolution. The less stable evolutions also show a worse overall result.

To further elaborate on that last point, we consider Fig. 2: It shows the average absolute value of change over 150-generations-wide windows of the $|\text{AF}(x) - \text{FPF}(x)|$ metric used in Fig. 2e. The plot was smoothed using a convolution kernel $\langle 1, \ldots, 1 \rangle$ of size 25. Roughly speaking, we can see the slope of the plots in Fig. 2e here. In this plot, good evolutionary processes should maintain rather low values according to Thesis 2. We can observe that Manhattan-diverse evolution maintains the lowest values almost throughout the entire evolution. While fitness sharing shows increases and decreases in matching the FPF at a higher level than Manhattan diversity, inherited fitness shows a huge spike in the beginning (as does the standard approach), thus making a much less stable match for the FPF. As proposed by Thesis 2, the match between AF and FPF roughly corresponds to the quality of the overall result of the evolutionary process.

As mentioned earlier, we also further analyzed the importance of the setting for λ for the evolution. Figure 3 shows the impact of λ on the best results generated by the evolution. $\lambda = 0$ equals the standard evolution in all previous plots. Unsurprisingly, we see that some amount of diversity-awareness improves the results of evolution but setting λ too high favors diversity over the actual objective fitness and thus yields very bad results. We want to add that more intricate version of Manhattan-based augmented fitness AF might aim to adjust the λ parameter during evolution just as inherited fitness and fitness sharing might want to adjust their parameters. For these experiments, we chose a static parameter setting for simplicity.

4.2 The route planning problem

The route planning problem is a discrete optimization problem with a similar motivation as the pathfinding problem. Again, we adapt the problem and its description from Gabor et al. (2018).

A robot needs to perform $n = 12$ different tasks in a fixed order by visiting relevant workstations. Each workstation can perform exactly one of the tasks and for each task, there are $o = 5$ identical workstations to choose from. Accordingly, a solution candidate is a vector $\langle w_1, \ldots, w_n \rangle$ with $w_i \in \{1, \ldots, o\}$ for all $1 \leq i \leq n$. See Fig. 4 for an illustration using a smaller setting. A single workstation W can be identified by a tuple of its task type and its number, i.e., $W = (i, k)$ for some $1 \leq i \leq n$ and $1 \leq k \leq o$. To mimic various means of transport, the distance $D(W, W')$ between every two workstations $W = (i, k)$ and $W' = (j, l)$ is randomized individually within a range of $[0, \frac{1}{n}] \subset \mathbb{R}$. Note that this (most likely) gives rise to a non-euclidean space the robot is navigating. The objective fitness for this minimization problem is given via

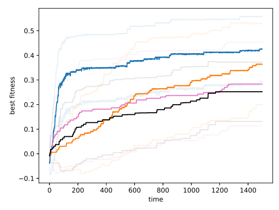
$$\text{OF}(\langle w_1, \ldots, w_n \rangle) = \sum_{i=1}^{n-1} D((i, w_i), (i+1, w_{i+1})). \quad (16)$$

Again, we from here also construct an augmented fitness $\text{AF}(x) = (1 - \lambda) \cdot \text{OF}(x) + \lambda \cdot \text{SF}(x)$ (cf. Definition 1) but now use the Hamming distance as a secondary fitness so that

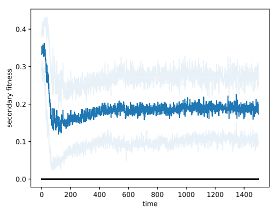
$$\text{SF}(x) = \frac{1}{2n} \cdot \text{avg}_{x' \in \sigma_4(X)} \mathit{hamming}(x, x') \quad (17)$$

Taken from original publication: Thomas Gabor, Thomy Phan, and Claudia Linnhoff-Popien. Productive fitness in diversity-aware evolutionary algorithms. *Natural Computing*, pages 1–14, 2021

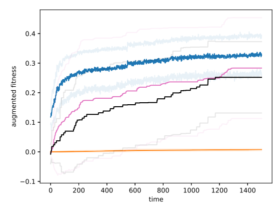
Fig. 2 Evolution for the pathfinding problem. Standard evolutionary process using OF shown in black, diversity-aware evolutionary process using AF with Manhattan distance shown in blue. Inherited fitness (purple) and fitness sharing (orange) shown for comparison. All results averaged over 25 independent runs, the standard deviation is shown in transparent lines



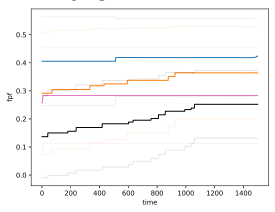
(a) Best objective fitness OF per generation.



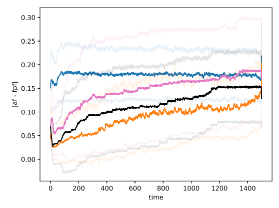
(b) Secondary fitness, i.e., diversity score, of the objectively best (according to OF) individual per generation.



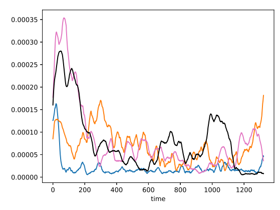
(c) Augmented fitness of the objectively best (according to OF) individual per generation.



(d) Best final productive fitness FPF per generation.



(e) Average per-individual difference between augmented fitness AF and final productive fitness FPF.



(f) Average amount of absolute change of per-individual difference between augmented fitness AF and final productive fitness FPF.

$$\text{where } hamming(\langle w_1, \ldots, w_n \rangle, \langle w'_1, \ldots, w'_n \rangle) = \sum_{i=1}^{n} h(w_i, w'_i) \tag{18}$$

$$\text{and } h(w, w') = \begin{cases} 0 & \text{if } w = w' \\ 1 & \text{otherwise.} \end{cases} \tag{19}$$

Besides, we apply the same evolutionary processes as in Sect. 4.1 but the parameter search shown in Fig. 5 now recommended $\lambda = 0.25$ for weighting now Hamming-based diversity.[6] We evolve 20 independent populations of size 50 for 400 generations and plot the same data we have seen before: Fig. 6a shows the best fitness achieved in evolution. Inherited fitness takes a lot more time but eventually almost reaches the level of Manhattan-diversity. However, both methods yield similarly solid results as fitness sharing or the naïve algorithm. This is mirrored by all methods showing quite stable behavior in Figs. 6e and 6f with the standard approach showing the highest fall within the first few generations as it matches FPF the least.

[6] It should be noted that the difference for various settings of diversity weights (including $\lambda = 0$) is much less pronounced for this domain as can be ssen in Fig. 5.

Taken from original publication: Thomas Gabor, Thomy Phan, and Claudia Linnhoff-Popien. Productive fitness in diversity-aware evolutionary algorithms. *Natural Computing*, pages 1–14, 2021

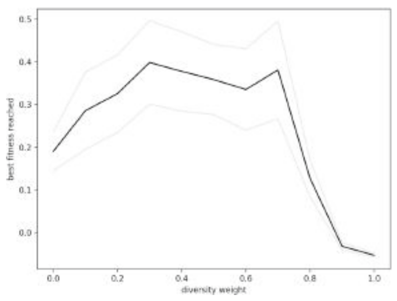


Fig. 3 Parameter analysis for the diversity weight λ. We show best fitness among all generations for 20 different settings of $\lambda = 0.0, 0.05, \ldots, 1.0$. All results averaged over 25 independent runs each, the standard deviation shown in transparent lines

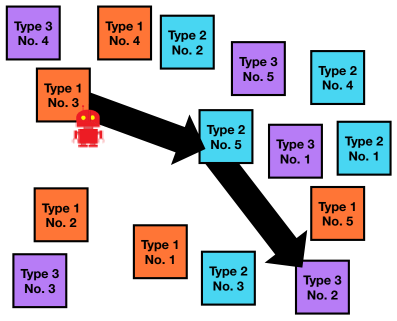


Fig. 4 Illustration of the route planning problem for $n = 3$ tasks and $o = 5$ workstations per task

However, the results for all evolutions are very close together for the route planning problem.

4.3 Schwefel

Finally, we consider one[7] of the canonical benchmark problems for evolutionary algorithms. The implementation of the Schwefel problem is taken from Rainville et al. (2012) while our study on the impact of diversity again follows experiments performed in Gabor et al. (2018).

The original fitness function is given as

[7] We also tested other famous problems like Rainville et al. (2012) with essentially similar results.

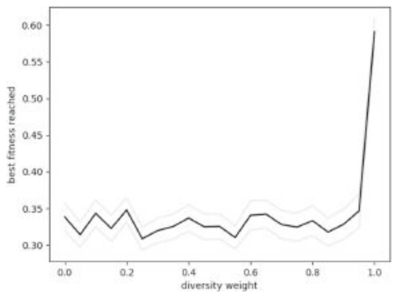


Fig. 5 Parameter analysis for the diversity weight λ for the route planning problem. We show best fitness among all generations for 20 different settings of $\lambda = 0.0, 0.05, \ldots, 1.0$. All results averaged over 20 independent runs each, the standard deviation shown in transparent lines

$$schwefel(\langle x_1, \ldots, x_n \rangle) = 418.9828872724339 \cdot n - \sum_{i=1}^{n} x_i \cdot \sin(\sqrt{|x_i|}) \quad (20)$$

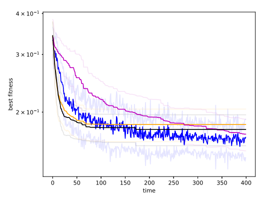
with $x_1, \ldots, x_n \in [-500; 500] \subset \mathbb{R}$ where $n = 8$ is the dimensionality we use in our experiments (Rainville et al. 2012).[8] The resulting function is illustrated in Fig. 7. The Schwefel problem is a minimization problem looking for an x so that $schwefel(x) = 0$. Again, we normalize the result values defining $\text{OF}(x) = \frac{1}{4000} \cdot schwefel(x)$. Manhattan distance uses diversity weight $\lambda = 0.3$ as suggested by Fig. 8.

We run the same kind of analysis as for the previous problems and plot the same data in Fig. 9. These experiments were performed on populations of size 50 evolving for 400 generations. Figure 9a shows a mixed picture: The Manhattan-diverse evolution again outperforms the standard approach, but inherited fitness hardly yields any benefit while fitness sharing performs worse than the standard approach. In Fig. 9d we see a different picture than before: All final productive fitness values are not as stable anymore but vary throughout the evolution, suggesting that solutions to the Schwefel problem are more influenced by migration (and thus show no continuous lineage) than for the other problems considered. Fig. 9e shows the $|\text{AF} - \text{FPF}|$ metric, which measures the individual difference between augmented fitness and final productive fitness. We observe very clearly that, after a short starting

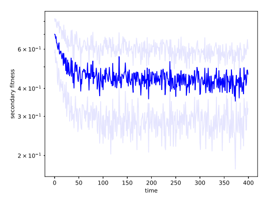
[8] As can be seen in Eq. 20, the Schwefel function is identical in each dimension. Choosing a higher-dimensional instance thus does not substantially change the nature of the fitness landscape. For ease of analysis, we settle for $n = 8$.

Taken from original publication: Thomas Gabor, Thomy Phan, and Claudia Linnhoff-Popien. Productive fitness in diversity-aware evolutionary algorithms. *Natural Computing*, pages 1–14, 2021

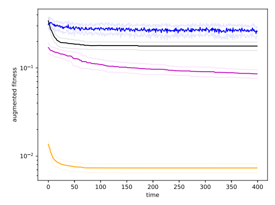
Fig. 6 Evolution for the route planning problem. Standard evolutionary process using OF shown in black, diversity-aware evolutionary process using AF with Hamming distance shown in blue. Inherited fitness (purple) and fitness sharing (orange) shown for comparison. Results averaged over 20 independent runs, the standard deviation is shown in transparent lines



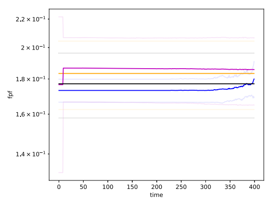
(a) Best objective fitness OF per generation



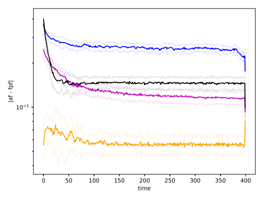
(b) Secondary fitness, i.e., diversity score, of the objectively best (according to OF) individual per generation.



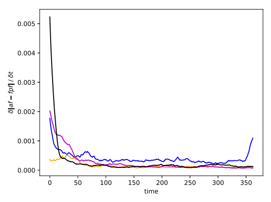
(c) Augmented fitness of the objectively best (according to OF) individual per generation.



(d) Best final productive fitness FPF per generation.



(e) Average per-individual difference between augmented fitness AF and final productive fitness FPF.



(f) Average amount of absolute change of per-individual difference between augmented fitness AF and final productive fitness FPF.

phase, this distance remains much more stable for the Manhattan-diverse evolution than any other approach, indicating that the augmented fitness is approximating the final productive fitness. Note again that in the last few generations, computing the final productive fitness has limited meaning. In Fig. 9f this behavior shows clearly as Manhattan-diverse evolution forms a line at the very bottom of the plot with almost no change in how AF matches FPF. All other approaches, which perform noticeably worse, also show a much more erratic pattern in how well their augmented fitness matches the final productive fitness.

5 Related work

Diversity has been a central topic of research in evolutionary algorithms (den Heijer and Eiben 2012; Morrison and De Jong 2001; Toffolo and Benini 2003; Ursem 2002). Its positive effect on the evolutionary process has often been observed there, but rarely been interpreted beyond a biological metaphor, i.e., "diversity is a key element of the biological theory of natural selection and maintaining high diversity is supposed to be generally beneficial" (Corno et al. 2005).

Taken from original publication: Thomas Gabor, Thomy Phan, and Claudia Linnhoff-Popien. Productive fitness in diversity-aware evolutionary algorithms. *Natural Computing*, pages 1–14, 2021

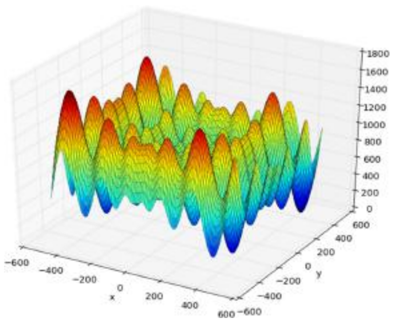


Fig. 7 Illustration of the Schwefel function for $n = 2$ dimensions. Image taken from (Benchmarks 2020)

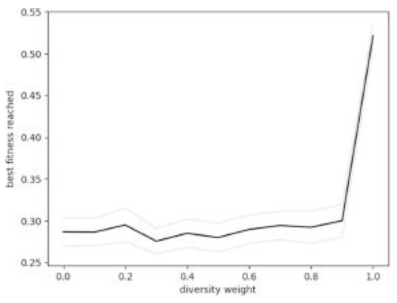


Fig. 8 Parameter analysis for the diversity weight λ for the Schwefel problem. We show best fitness among all generations for 10 different settings of $\lambda = 0.0, 0.1, \ldots, 1.0$. All results averaged over 20 independent runs each, the standard deviation shown in transparent lines

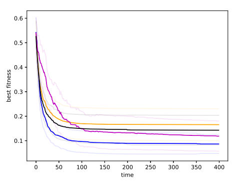
Without much concept of what to look for in a mechanism for diversity-awareness, lots of variants have spawned in research. Instead of repeating them, we would like to point out a few resources for a comprehensive overview: Burke et al. (2004), among others like Brameier and Banzhaf (2002 or McPhee and Hopper (1999) discuss various means to measure and promote diversity in genetic programming, which for the most part should apply to all evolutionary algorithms. They also provide an extensive analysis of the connection between diversity and achieved fitness, but do not define productive fitness or a similar notion. A more recent comprehensive overview of means to describe and enable diversity has been put together by Squillero and Tonda (2016), also providing a taxonomy on various classes of approaches to diversity. Gabor et al. (2018) provide a quantitative analysis of various means of maintaining inheritance-based diversity on standard domains like the ones we used in this paper. Regarding the multitude of diversity mechanisms present in research, however, it is most important to also point out the results of Wineberg and Oppacher (2003), who most drastically show that "all [notions of diversity] are restatements or slight variants of the basic sum of the distances between all possible pairs of the elements in a system" and suggest that "experiments need not be done to distinguish between the various measures", a point which we already built upon in our evaluation.

Note that a variety of "meta-measurements" for the analysis of evolutionary processes exist: *Effective fitness* measures the minimum fitness required for an individual to increase in dominance at a given generation (when in competition with the other individuals) (Stephens 1999). It is related to *reproductive fitness*, which is the probability of an individual to successfully produce offspring (Hu and Banzhaf 2010). Both occur at the foundation of productive fitness, but do not include the (computationally overly expensive) diachronical analysis of the overall effect for the end result. Our approach is also comparable to *entropy-based diversity preservation* (Squillero and Tonda 2008), where the positive effect of certain individuals on the population's entropy is measured and preserved in order to deliberately maintain higher entropy levels. By contrast, our approach is based on the fitness values only (without the need to look into the individuals beyond their genealogical relationships) and thus also cannot be used directly as a secondary goal in evolution but purely as a tool of a posteriori analysis on the effectiveness of other secondary goal definitions.

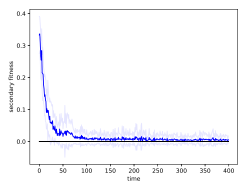
When we construct the "optimal evolutionary process", we construct a dynamic optimization problem from a traditionally static one. It is interesting that specifically dynamic or *on-line* (Bredeche et al. 2009) evolutionary algorithms have been shown to benefit from increased diversity especially when facing changes in their fitness functions (Gabor et al. 2018; Grefenstette 1992). While this is obviously intuitive as more options in the population allow for higher coverage of possible changes, the reverse connection (pointed to by this work) is not stated there, i.e., that diversity in static domains may work because even for static domains the optimization process is inherently dynamic to some degree.

Taken from original publication: Thomas Gabor, Thomy Phan, and Claudia Linnhoff-Popien. Productive fitness in diversity-aware evolutionary algorithms. *Natural Computing*, pages 1–14, 2021

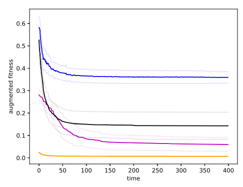
Fig. 9 Evolution for the Schwefel problem. Standard evolutionary process using OF shown in black, diversity-aware evolutionary process using AF with Manhattan distance shown in blue. Inherited fitness (purple) and fitness sharing (yellow) shown for comparison. Results averaged over 20 independent runs, the standard deviation is shown in transparent lines



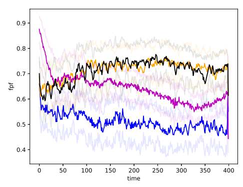
(a) Best objective fitness OF per generation



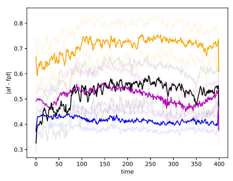
(b) Secondary fitness, i.e., diversity score, of the objectively best (according to OF) individual per generation.



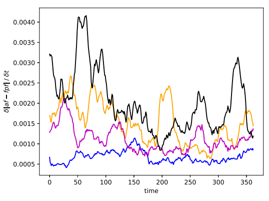
(c) Augmented fitness of the objectively best (according to OF) individual per generation.



(d) Best final productive fitness FPF per generation.



(e) Average per-individual difference between augmented fitness AF and final productive fitness FPF.



(f) Average amount of absolute change of per-individual difference between augmented fitness AF and final productive fitness FPF.

6 Conclusion

We have introduced the novel notion of final productive fitness FPF (and all the definitions it is built upon). We make a theoretical argument that FPF is the goal an optimal evolutionary process should strive for to achieve the best overall results. However, FPF cannot be computed efficiently in advance, producing the need for an approximation. We argue that the well-known technique of augmenting the objective fitness function with an additional diversity goal (when it helps) happens to effectively approximate the theoretically derived FPF (at least better than just the objective fitness on its own). We have shown this connection empirically on benchmark domains. We argue that this provides first insight into *why* and *how* diversity terms are beneficial to evolutionary processes.

Immediate future work would consist of answering the *when* and *which*: We have tested several domains for evolutionary algorithms and many are too simple to further benefit from explicit diversity-awareness. Maybe FPF can be used to derive a criterion to estimate the usefulness of diversity in advance. Similarly, many mechanism to cater explicitly to diversity exist. While many can be subsumed by the pairwise distance used here (Wineberg and

Oppacher 2003), others may still show different behavior. Their relation to FPF requires further work.

There are means of maintaining diversity without altering the fitness function, most prominently structural techniques like island models (Tomassini 2006; Whitley et al. 1999) or hypermutation phases (Morrison and De Jong 2000; Simões and Costa 2002). As no match for $|\text{AF} - \text{FPF}|$ can be computed for them, we omitted them in this first analysis. Final productive fitness may be a useful tool to translate these structural means into an effect of the fitness function.

Eventually, there may be even more direct or outright better (compared to using diversity or similarly augmented fitness) approximations of FPF to be found now that we know what we are looking for. The ultimate goal of the research into FPF might be to utilize it directly or indirectly in actually constructing new types of evolutionary algorithms (instead of it "only" helping to explain how well-known types work). We can imagine:

- We could compute FPF for a simplified version of the problem (and the algorithm) for various parametrizations (Eiben and Smith 2003; Mitchell 1998). The FPF's values could then help evaluate the respective parametrization's success. However, for most complex problems it is not entirely clear how to derive simpler instances that still preserve the interesting or challenging aspects of their larger counterparts. Furthermore, it is unclear how FPF might provide more information than just using the respective runs' OF.
- In this paper, we checked how well various established models for fitness functions approximate FPF. Instead, we might now construct new models with the goal to approximate FPF better. Surrogate models have been used in evolutionary algorithms to approximate computationally expensive fitness functions (Gabor and Altmann 2019; Jin 2005; Jin and Sendhoff 2004). In our case, a surrogate model would have to approximate our a-posteriori-approximation of FPF and could then maybe save time for future evaluations. It should be noted that our results make it seem plausible that no general model for FPF on all domains should exist and we cannot train the surrogate as the algorithm goes along since our target metric is only computed a posteriori. However, we might still be able to learn an n-PF surrogate for small n or a similar target metric. In Gabor and Linnhoff-Popien (2020) we already suspect (based on the a posteriori approximation as we do in this paper) that FPF constructs a simpler fitness landscape compared to, quite like surrogates do, suggesting that surrogates may be trained to achieve a similar fitness landscape. That dynamic should be explored in future work.
- In Gabor and Belzner (2017) and Gabor et al. (2018) we introduce *genealogical distance* as a diversity metric for evolutionary algorithms. We show that it provides similar although at times inferior results to distance-based diversity metrics (i.e., the Manhattan and Hamming diversity we use in this paper). However, genealogical diversity may provide means to approximate genealogical relations between individuals making our a-posteriori-approximation much easier to compute (probably at the expense of accuracy). Future work should evaluate if that approach brings any relevant benefits.

Some of these may be applicable to other methods of optimization as well: We suspect that the notion of final productive fitness translates directly to all optimization methods (ranging from simulated annealing or particle swarm optimization to Monte-Carlo tree search and back-propagation) that may or may not already implement means to approximate final productive fitness rather than just objective fitness.

Acknowledgements We thank the anonymous reviewers for their in-depth comments and their contribution to this paper. We also thank Lenz Belzner for intense discussions that helped make this paper possible.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

Benchmarks – DEAP 1.3.1 documentation. http://deap.readthedocs.io/en/master/api/benchmarks.html. Accessed: 2020-04-30

Brameier M, Banzhaf W (2002) Explicit control of diversity and effective variation distance in linear genetic programming. In: European Conference on Genetic Programming, pp 37–49. Springer

Bredeche N, Haasdijk E, Eiben A (2009) On-line, on-board evolution of robot controllers. In: International Conference on Artificial Evolution (Evolution Artificielle), pp 110–121. Springer

Brown G, Wyatt J, Harris R, Yao X (2005) Diversity creation methods: a survey and categorisation. Inf Fusion 6(1):5–20

Taken from original publication: Thomas Gabor, Thomy Phan, and Claudia Linnhoff-Popien. Productive fitness in diversity-aware evolutionary algorithms. *Natural Computing*, pages 1–14, 2021

Burke EK, Gustafson S, Kendall G (2004) Diversity in genetic programming: an analysis of measures and correlation with fitness. IEEE Trans Evolut Comput 8(1):47–62

Chen JH, Goldberg DE, Ho SY, Sastry K (2002) Fitness inheritance in multi-objective optimization. In: Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation, pp 319–326

Conti E, Madhavan V, Such FP, Lehman J, Stanley K, Clune J (2018) Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. In: Advances in Neural Information Processing Systems, pp 5027–5038

Corno F, Sánchez E, Squillero G (2005) Evolving assembly programs: how games help microprocessor validation. IEEE Trans Evolut Comput 9(6):695–706

Črepinšek M, Liu SH, Mernik M (2013) Exploration and exploitation in evolutionary algorithms: a survey. ACM Comput Surv 45(3):35

Eiben AE, Smith JE et al (2003) Introduction to evolutionary computing, vol 53. Springer, Berlin

Gabor T, Altmann P (2019) Benchmarking surrogate-assisted genetic recommender systems. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, pp 1568–1575

Gabor T, Belzner L (2017) Genealogical distance as a diversity estimate in evolutionary algorithms. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, pp 1572–1577

Gabor T, Belzner L, Linnhoff-Popien C (2018) Inheritance-based diversity measures for explicit convergence control in evolutionary algorithms. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp 841–848. ACM

Gabor T, Belzner L, Phan T, Schmid K (2018) Preparing for the unexpected: Diversity improves planning resilience in evolutionary algorithms. In: 2018 IEEE International Conference on Autonomic Computing (ICAC), pp 131–140. IEEE

Gabor T, Linnhoff-Popien C (2020) A formal model for reasoning about the ideal fitness in evolutionary processes. In: T Margaria, B Steffen (eds) Leveraging Applications of Formal Methods, Verification and Validation: Engineering Principles - 9th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2020, Rhodes, Greece, October 20-30, 2020, Proceedings, Part II, *Lecture Notes in Computer Science*, vol. 12477, pp 473–490. Springer. https://doi.org/10.1007/978-3-030-61470-6_28

Grefenstette JJ et al (1992) Genetic algorithms for changing environments. PPSN 2:137–144

Hart E, Sim K (2018) On constructing ensembles for combinatorial optimisation. Evolut Comput 26(1):67–87

den Heijer E, Eiben A (2012) Maintaining population diversity in evolutionary art. In: International Conference on Evolutionary and Biologically Inspired Music and Art, pp 60–71. Springer

Hu T, Banzhaf W (2010) Evolvability and speed of evolutionary algorithms in light of recent developments in biology. J Artif Evolut Appl 2010:1

Jin Y (2005) A comprehensive survey of fitness approximation in evolutionary computation. Soft Comput 9(1):3–12

Jin Y, Sendhoff B (2004) Reducing fitness evaluations using clustering techniques and neural network ensembles. In: Genetic and Evolutionary Computation Conference, pp 688–699. Springer

McPhee NF, Hopper NJ (1999) Analysis of genetic diversity through population history. In: Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 2, pp 1112–1120. Morgan Kaufmann Publishers Inc

Mitchell M (1998) An introduction to genetic algorithms. MIT press, Cambridge

Morrison RW, De Jong KA (2000) Triggered hypermutation revisited. In: Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No. 00TH8512), vol 2, pp 1025–1032. IEEE

Morrison RW, De Jong KA (2001) Measurement of population diversity. In: International Conference on Artificial Evolution (Evolution Artificielle), pp 31–41. Springer

Ng AY, Harada D, Russell S (1999) Policy invariance under reward transformations: theory and application to reward shaping. ICML 99:278–287

Pugh JK, Soros LB, Stanley KO (2016) Quality diversity: a new frontier for evolutionary computation. Front Robot AI 3:40

Rainville D, Fortin FA, Gardner MA, Parizeau M, Gagné C et al. (2012) Deap: A python framework for evolutionary algorithms. In: Proceedings of the 14th annual conference companion on Genetic and evolutionary computation, pp 85–92. ACM

Sareni B, Krahenbuhl L (1998) Fitness sharing and niching methods revisited. IEEE Trans Evolut Comput 2(3):97–106

Simões A, Costa E (2002) Using genetic algorithms to deal with dynamic environments: a comparative study of several approaches based on promoting diversity. Proceed Genet Evolut Comput Conf GECCO 2:698–707

Squillero G, Tonda A (2016) Divergence of character and premature convergence: a survey of methodologies for promoting diversity in evolutionary optimization. Inf Sci 329:782–799

Squillero G, Tonda AP (2008) A novel methodology for diversity preservation in evolutionary algorithms. In: Proceedings of the 10th annual conference companion on Genetic and evolutionary computation, pp 2223–2226. ACM

Stephens CR (1999) "Effective" fitness landscapes for evolutionary systems. In: Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406), vol 1, pp 703–714. IEEE

Toffolo A, Benini E (2003) Genetic diversity as an objective in multi-objective evolutionary algorithms. Evolut Comput 11(2):151–167

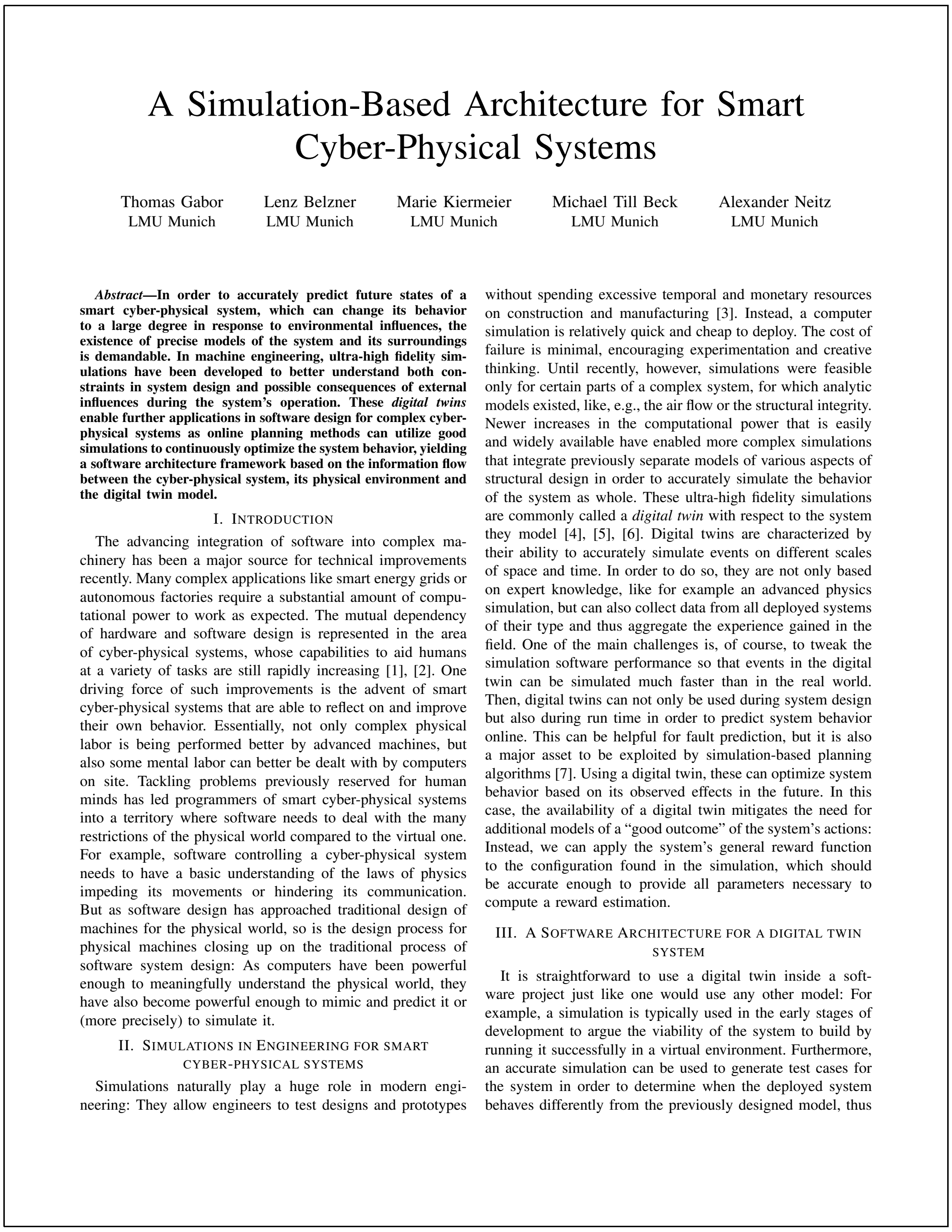
Tomassini M (2006) Spatially structured evolutionary algorithms: artificial evolution in space and time. Springer, Berlin

Ursem RK (2002) Diversity-guided evolutionary algorithms. In: International Conference on Parallel Problem Solving from Nature, pp 462–471. Springer

Whitley D, Rana S, Heckendorn RB (1999) The island model genetic algorithm: on separability, population size and convergence. J Comput Inf Technol 7(1):33–47

Wineberg M, Oppacher F (2003) The underlying similarity of diversity measures used in evolutionary computation. In: Genetic and Evolutionary Computation Conference, pp 1493–1504. Springer

Taken from original publication: Thomas Gabor, Lenz Belzner, Marie Kiermeier, Michael Till Beck, and Alexander Neitz. A simulation-based architecture for smart cyber-physical systems. In *The International Workshop on Models@run.time for Self-Aware Computing Systems*, 2016



A Simulation-Based Architecture for Smart Cyber-Physical Systems

Thomas Gabor, LMU Munich
Lenz Belzner, LMU Munich
Marie Kiermeier, LMU Munich
Michael Till Beck, LMU Munich
Alexander Neitz, LMU Munich

***Abstract*—In order to accurately predict future states of a smart cyber-physical system, which can change its behavior to a large degree in response to environmental influences, the existence of precise models of the system and its surroundings is demandable. In machine engineering, ultra-high fidelity simulations have been developed to better understand both constraints in system design and possible consequences of external influences during the system's operation. These *digital twins* enable further applications in software design for complex cyber-physical systems as online planning methods can utilize good simulations to continuously optimize the system behavior, yielding a software architecture framework based on the information flow between the cyber-physical system, its physical environment and the digital twin model.**

I. Introduction

The advancing integration of software into complex machinery has been a major source for technical improvements recently. Many complex applications like smart energy grids or autonomous factories require a substantial amount of computational power to work as expected. The mutual dependency of hardware and software design is represented in the area of cyber-physical systems, whose capabilities to aid humans at a variety of tasks are still rapidly increasing [1], [2]. One driving force of such improvements is the advent of smart cyber-physical systems that are able to reflect on and improve their own behavior. Essentially, not only complex physical labor is being performed better by advanced machines, but also some mental labor can better be dealt with by computers on site. Tackling problems previously reserved for human minds has led programmers of smart cyber-physical systems into a territory where software needs to deal with the many restrictions of the physical world compared to the virtual one. For example, software controlling a cyber-physical system needs to have a basic understanding of the laws of physics impeding its movements or hindering its communication. But as software design has approached traditional design of machines for the physical world, so is the design process for physical machines closing up on the traditional process of software system design: As computers have been powerful enough to meaningfully understand the physical world, they have also become powerful enough to mimic and predict it or (more precisely) to simulate it.

II. Simulations in Engineering for smart cyber-physical systems

Simulations naturally play a huge role in modern engineering: They allow engineers to test designs and prototypes without spending excessive temporal and monetary resources on construction and manufacturing [3]. Instead, a computer simulation is relatively quick and cheap to deploy. The cost of failure is minimal, encouraging experimentation and creative thinking. Until recently, however, simulations were feasible only for certain parts of a complex system, for which analytic models existed, like, e.g., the air flow or the structural integrity. Newer increases in the computational power that is easily and widely available have enabled more complex simulations that integrate previously separate models of various aspects of structural design in order to accurately simulate the behavior of the system as whole. These ultra-high fidelity simulations are commonly called a *digital twin* with respect to the system they model [4], [5], [6]. Digital twins are characterized by their ability to accurately simulate events on different scales of space and time. In order to do so, they are not only based on expert knowledge, like for example an advanced physics simulation, but can also collect data from all deployed systems of their type and thus aggregate the experience gained in the field. One of the main challenges is, of course, to tweak the simulation software performance so that events in the digital twin can be simulated much faster than in the real world. Then, digital twins can not only be used during system design but also during run time in order to predict system behavior online. This can be helpful for fault prediction, but it is also a major asset to be exploited by simulation-based planning algorithms [7]. Using a digital twin, these can optimize system behavior based on its observed effects in the future. In this case, the availability of a digital twin mitigates the need for additional models of a "good outcome" of the system's actions: Instead, we can apply the system's general reward function to the configuration found in the simulation, which should be accurate enough to provide all parameters necessary to compute a reward estimation.

III. A Software Architecture for a digital twin system

It is straightforward to use a digital twin inside a software project just like one would use any other model: For example, a simulation is typically used in the early stages of development to argue the viability of the system to build by running it successfully in a virtual environment. Furthermore, an accurate simulation can be used to generate test cases for the system in order to determine when the deployed system behaves differently from the previously designed model, thus

Taken from original publication: Thomas Gabor, Lenz Belzner, Marie Kiermeier, Michael Till Beck, and Alexander Neitz. A simulation-based architecture for smart cyber-physical systems. In *The International Workshop on Models@run.time for Self-Aware Computing Systems*, 2016

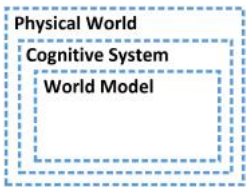


Fig. 1. Classical view of a simulation architecture: The world model used for the simulation is embedded inside the software system and controlled by a planning agent, for example.

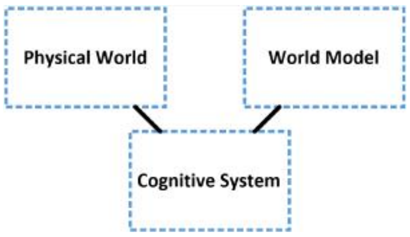


Fig. 2. The digital twin view of a simulation architecture: The cyber-physical system communicates with both its real world hardware and its simulation model using the same interface.

finding errors either in the simulation or the deployed system. It is clear that disposing of a high-fidelity simulation like a digital twin is quite beneficiary for these cases as well [4]. However, the digital twin not only brings a quantitative advantage in the form of improved test accuracy or fault detection, but it also features a unique qualitative advantage with regard to optimization and planning algorithms: Since the digital twin closely mimicks the physical world down to the smallest scales, it is also able to produce the same class of sensory input as the real world would produce and execute the same actions as the cyber-physical system's real actuators would execute in the physical world. This allows for a paradigm shift when it comes to integrating the logical model with the software architecture, as shown in Figures 1 and 2: Because the digital twin is able to implement the same interface as the sensors and actuators (or motors) used to interact with the physical world, it is possible to use the exact same action abstractions both for planning in the virtual space and for the execution of plans in the real space. Thus, from a software point of view, the sensorimotor controller connecting the cyber-physical system to its hardware components and the respective interface of the digital twin can be exchanged at will. Essentially, the difference between the physical world and the virtual world becomes transparent to the programmer of a smart cyber-physical system. The effort required in order to employ a digital twin is relatively high from a software engineer's perspective: High-fidelity simulations require both a huge amount of computational power to be evaluated during run time but usually also a huge amount of human labor to be developed. However, as previously discussed, such models are currently being developed by engineers in other fields in order to support design studies and machine maintenance. And the cost of computational power is still decreasing with technological advancements in hardware. On the other hand, having a simulation that can be interfaced with like the real world not only makes for a more elegant programming architecture, but mainly allows to test said simulation against real observations (and thus improve it) and allows for a general definition on how to integrate multiple world models that is then compatible with any version of a digital twin which may be plugged into the system.

A. Integrating and Using a Digital Twin

Figure 3 shows an instance model of a cyber-physical system. The Controller is the component that represents the interface to the physical world and thus the system's environment. It understands a predefined set of actions that can be executed on the physical actuators with the *sendAction* method. Event objects are generated by the Controller whenever it observes an update to its state, usually initiated by the system's sensors but also by the system's clock or other internal parts. Other components can subscribe to these events and are subsequently notified through the Controller's *publishTo* method. Commonly, the events observed directly through changes in sensor input are too subtle to be immediately useful to adjust the system's behavior. Thus, an *Aggregator* can be used to watch all the subtle events and abstract from the loads of data observed. In turn, an Aggregator can also publish Event objects to its subscribed observers but will usually be expected to do so less frequently.

The Cognitive System consists of components that basically react on Event objects by sending Action objects to the Controller. This may happen almost immediately, in which case we call the component an "intuitive" reactor meaning that the component is quick to compute a reaction to incoming events. However, the component may also first trigger a planning phase in order to compute a suitable reaction. The latter case is more interesting as it allows the system to behave more intelligently and is ultimately the main reason for variance and flexibility in system behavior in our smart cyber-physical system. There are multiple ways to implement planning into a cyber-physical system, but using the existing digital twin, it is natural to resort to *simulation-based planning*, i.e. planning methods relying on a model describing the whole environment from which test runs can be sampled.

One of the defining aspects of using simulation-based planning is that the simulation at work, in this case the digital twin, supports the same kinds of observations, i.e. events, and actions as the interface to the real world (represented by the Controller). In Figure 3, this shows as the digital twin component supports the *publishTo* and *sendAction* methods.

Taken from original publication: Thomas Gabor, Lenz Belzner, Marie Kiermeier, Michael Till Beck, and Alexander Neitz. A simulation-based architecture for smart cyber-physical systems. In *The International Workshop on Models@run.time for Self-Aware Computing Systems*, 2016

However, there is one main difference between the interfaces of the Controller and the digital twin: The flow of time can be controlled in the digital twin. This includes the fact that simulations in the digital twin can be started and halted by another software component, mainly the Planner, which is ultimately what makes the simulation useful for planning.[1] Furthermore, it is possible and often also quite advisable for a cyber-physical system to feature multiple digital twins. These may focus on different aspects of the environment and thus be used in different situations or may correspond to various internal components of the cyber-physical system. One may also instantiate multiple digital twins dynamically to account for uncertainty of the system about its own state: For example, when it is unclear which ones of several kind of motors are actually used in a factory robot, it may be the easiest way to simulate all possible kinds in order to avoid any risks. As discussed previously, the cyber-physical system may learn which motor it possesses by comparing the simulations from several digital twins to the observations made in the physical world over a certain period of time.

B. *Defining an Architectural Framework for Digital Twins*

Note that the instance diagram in Figure 3 is only an example of one possible configuration. In different cyber-physical systems, we would expect a different range and quantity of components. Typically, many different levels of intuitive reactors and planning components are combined to cover a wide range of possible situations and events. However, it is possible to generalize the architecture displayed in Figure 3; the resulting class diagram is shown in Figure 4. Both the Controller and the Digital Twin class inherit from the World superclass. This is what defines their ability to receive actions and produce events. On a more general level, every World can be regarded as an Event Source, to which any Event Observer can subscribe, thus forming a classic observer pattern [8]. It should be noted that Aggregators function as both Event Sources and Event Observers. Thus, multiple Aggregators can be constructed so that they form a hierarchy of information abstraction. For example, the Controller interacting with the physical world may publish events for every time its temperature sensor returns a reading of the current temperature. Due to random external circumstances (air flow, position relative to the sun, etc.), these readings may vary by some degrees even in a relatively short amount of time. In some scenarios, it may only be critical for the cyber-physical system to notice if the temperature consistently increases by several degrees over a longer period of time. An aptly configured Aggregator might thus observe every little event thrown by the temperature sensor (as part of the Controller) but only generate an event of its own when it notices the described pattern in temperature data. Other components of the system may then only subscribe to the Aggregator instead of the temperature sensor to hide superfluous information from their event interface. Lastly, Cognitive System is the broad term for any component that can observe events and produce actions, i.e. react to some kind of impulse. As previously mentioned, these components may come in different degrees of complexity. The simpler ones are called Intuitive Reactor while a Planner is able to produce more flexible behavior by optimizing its actions according to certain goals defined by the programmer of the system. To do so, the Planner is supposed to use the digital twin to predict the success of its plans before choosing a single plan to execute.

[1]Note that while this may seem a trivial property to achieve when considering simpler models, for sufficiently complex world models, there often is an upper limit as to what speed-up of time with respect to the physical world can be achieved while keeping the level of detail constant. Effectively, we may not have arbitrary control over time in the digital twin because we have limited computational resources.

Tier	Description
0	**physical necessity**: laws of nature
1	**machine-environment interface**: circuit-level control, sensor/actuator hardware, computational capabilties
2	**immediate reaction**: watchdogs, fixed behavioral rulesets, expert systems
3	**planned reaction**: reward functions for online planners and self-awareness

Fig. 5. Overview over the different levels of control present in the information flow of a cyber-physical system.

IV. Safety Engineering using a Simulation-Based Architecture

From an engineering point of view, the architectural framework presented in Figure 4 is on the more general side. It is supposed to be abstract enough so that it can be applied to a wide range of challenges for which smart cyber-physical systems are being developed. The central focus point of the proposed architecture is information flow: All behavior exerted by the cyber-physical system is triggered by some kind of event. Essentially, behavior is nothing more than a translation from observations to actions.

When dealing with complex smart cyber-physical systems, it is often difficult to predict in advance how exactly the system will behave. However, for basically all practical applications there need to be certain guarantees on the behavior of the system as it would be unsafe to deploy otherwise. Focusing on information flow helps us identify different levels of control present in the simulation-based architectural model. We can thus group different methods of exerting control on the behavior of a smart cyber-physical system into different tiers. These can be seen as different classes of constraints we can have our smart cyber-physical system comply to. For a quick overview over the classes we are about to discuss, refer to the table in Figure 5.

A. *Physical Necessity (Tier 0)*

Starting with tier 0, we are facing the odd candidate of these levels of control: In the most basic sense, the laws of the physical world the system operates in dictate a certain level of control. Our smart cyber-physical system is never free

Taken from original publication: Thomas Gabor, Lenz Belzner, Marie Kiermeier, Michael Till Beck, and Alexander Neitz. A simulation-based architecture for smart cyber-physical systems. In *The International Workshop on Models@run.time for Self-Aware Computing Systems*, 2016

to decide on actions that would not comply with the laws of physics, for example. Obviously, this seems like a rather trivial rule to model in our classification of control mechanisms, but it is not be neglected in the simulation part of the system. Since the digital twin is supposed to closely model reality, we have to explicitly model tier 0 control mechanisms for our simulation, even if they do not seem to directly temper with our planning process; we would not have to do this when using more abstract system models for planning. Thus, the need to accurately describe tier 0 constraints on system behavior is a unique consequence of using a high-fidelity simulation. Furthermore, there is no way to alter these control mechanisms when designing a system so the system developers need to understand and cope with what is preset by nature. Luckily, the phenomona of the physical world are usually well understood on the scales contemporary machines are dealing with and have been subject to simulation numerous times before.

B. Machine-Environment Interface (Tier 1)

Tier 1 summarizes control mechanisms placed at the system's interface with the environment. They are not enforced by anything but the hardware itself, but they can be regarded as part of the system's physical "body" to some extent. In contrast to tier 0 control, these mechanisms are usually created by the system developers deliberately. A typical example for a tier 1 control mechanism would be a circuit preventing a motor part from overheating by shutting down the engine. Many of these feedback loops are usually directly built into hardware and not exposed to software at all. This means that they cannot be altered online at run time, but are usually fixed once the respective parts are built. A rather delicate variant of this kind of control may happen with sensors and actuators: When a sensor can, for example, only provide a certain quality of data or only measure certain inputs, that can result in a severe constraint on the behavior of the cyber-physical system as it can no longer discern all different states of the physical world and is thus forced to treat situations similarily when they yield the same sensor data. The same effect can occur when regarding the precision with which actuators can comply to an action specification sent to them. In either case, it is clearly important to consider endowing the system with the necessary hardware so that it can reach solutions to all problems desired to be solved by the system. From a software architect's point of view, tier 1 control mechanisms mainly play a role when programming the Controller operating all the sensors and actuators.[2] However, since software has no means of altering tier 1 control dynamically, it must be informed about the tier 1 mechanisms by having them included in the digital twin's simulations. Thus, tier 1 control needs to be placed both at the interface with the real world, but also at the interface with the virtual world.

[2]While we have previously discussed issues on the hardware side, the Controller needs to accurately expose the level of precision in sensor and actor data that it can actually deliver and is thus usually dependend on the hardware design process as well. Also consider a system in which sensors produce loads of highly accurate data but the deployed software system is unable to process it because the amounts of computational power required would exceed the capacity the system's CPU, or worse, any CPU on the planet.

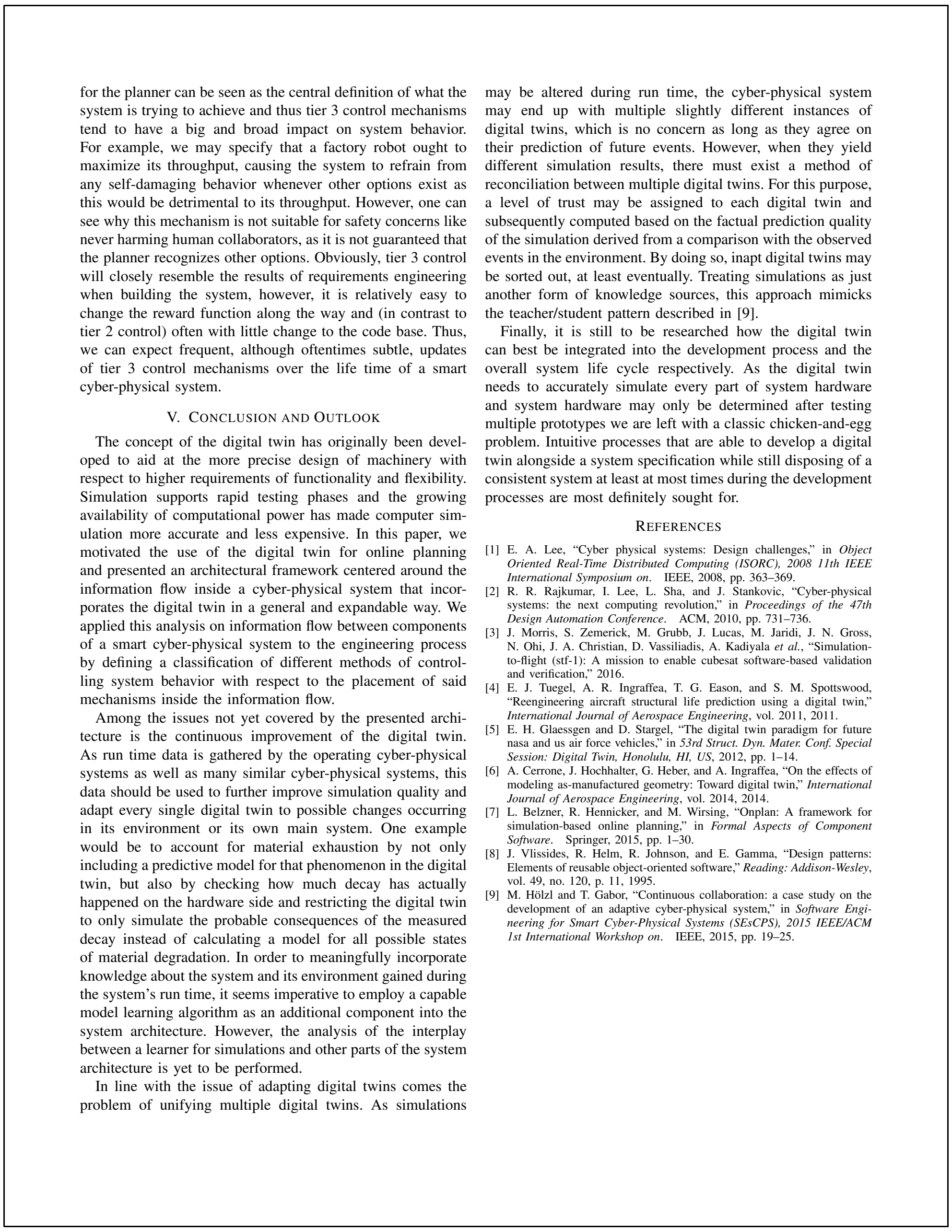
C. Immediate Reaction (Tier 2)

Tier 2 corresponds closely to the Intuitive Reactor in our system architecture. A classic example of tier 2 control would be to equip a self-driving factory robot with a software component that constantly checks the sensor events with respect to whether there is a human worker walking in the close vicinity. In that case, the Intuitive Reactor would cause the robot to stop by issuing the respective action. We expect the robot to do so immediately regardless of what it currently tries to achieve as once the Intuitive Reactor recognizes one of the situations it is looking out for, it is expected to know how to react correctly without weighing different options too much (hence "*intuitive* reactor"). Software components like these are also called *watchdogs* and we expect any sufficiently complex cyber-physical system to feature many of those. If situations may occur in which multiple Intuitive Reactors may trigger, it is up to the system designers to specify a reasonable order of precedence amongst them. In contrast to the lower tiers of control, tier 2 control mechanisms are part of the cyber-physical system's main information flow cycle. Thus, in order for them to work they depend on an at least rudimentally healthy system, which is also an argument why certain crucial control mechanisms should be placed at tier 1 when possible. However, tier 2 can also use the full set of features available to the system, usually including a variety of sensors and abstraction layers on observed data, i.e. Aggregators, which should aid them to make their decisions more precise. As Intuitive Reactors are completely deployed as software components, they are susceptible to online updates and may be adjusted during run time. For the system developer, it also follows that the system is not stuck with all the watchdogs it is deployed with and watchdogs can be removed if they no longer fit a new task the system is expected to fulfill or new ones can be added if a new tendency for dangerous behavior has been discovered, for example because lower or higher tier control mechanisms failed to work as expected in practical situations.

D. Planned Reaction (Tier 3)

The highest tier of control in a smart cyber-physical system results in a *planned* or *deliberate reaction* as opposed to an intuitive one. Tier 3 control mechanisms are carried out by the Planner or multiple Planners active in the cyber-physical system. Typically, tier 3 control is implemented by defining a reward function which the Planners try to optimize. When the Planners work successfully, this results in the system showing a tendency to behave in a way that maximizes the value of the reward function. Both hard and soft constraints can be implemented by defining the reward function accordingly. However, planners for sufficiently complex problems usually function non-deterministically and thus tier 3 control is always "a bit fuzzy" in nature, implying that it is mostly uncertain to which extent a set constraint will be fulfilled at a specific point in time. Nonetheless, in a smart system, the reward function

Taken from original publication: Thomas Gabor, Lenz Belzner, Marie Kiermeier, Michael Till Beck, and Alexander Neitz. A simulation-based architecture for smart cyber-physical systems. In *The International Workshop on Models@run.time for Self-Aware Computing Systems*, 2016



for the planner can be seen as the central definition of what the system is trying to achieve and thus tier 3 control mechanisms tend to have a big and broad impact on system behavior. For example, we may specify that a factory robot ought to maximize its throughput, causing the system to refrain from any self-damaging behavior whenever other options exist as this would be detrimental to its throughput. However, one can see why this mechanism is not suitable for safety concerns like never harming human collaborators, as it is not guaranteed that the planner recognizes other options. Obviously, tier 3 control will closely resemble the results of requirements engineering when building the system, however, it is relatively easy to change the reward function along the way and (in contrast to tier 2 control) often with little change to the code base. Thus, we can expect frequent, although oftentimes subtle, updates of tier 3 control mechanisms over the life time of a smart cyber-physical system.

V. Conclusion and Outlook

The concept of the digital twin has originally been developed to aid at the more precise design of machinery with respect to higher requirements of functionality and flexibility. Simulation supports rapid testing phases and the growing availability of computational power has made computer simulation more accurate and less expensive. In this paper, we motivated the use of the digital twin for online planning and presented an architectural framework centered around the information flow inside a cyber-physical system that incorporates the digital twin in a general and expandable way. We applied this analysis on information flow between components of a smart cyber-physical system to the engineering process by defining a classification of different methods of controlling system behavior with respect to the placement of said mechanisms inside the information flow.

Among the issues not yet covered by the presented architecture is the continuous improvement of the digital twin. As run time data is gathered by the operating cyber-physical systems as well as many similar cyber-physical systems, this data should be used to further improve simulation quality and adapt every single digital twin to possible changes occurring in its environment or its own main system. One example would be to account for material exhaustion by not only including a predictive model for that phenomenon in the digital twin, but also by checking how much decay has actually happened on the hardware side and restricting the digital twin to only simulate the probable consequences of the measured decay instead of calculating a model for all possible states of material degradation. In order to meaningfully incorporate knowledge about the system and its environment gained during the system's run time, it seems imperative to employ a capable model learning algorithm as an additional component into the system architecture. However, the analysis of the interplay between a learner for simulations and other parts of the system architecture is yet to be performed.

In line with the issue of adapting digital twins comes the problem of unifying multiple digital twins. As simulations may be altered during run time, the cyber-physical system may end up with multiple slightly different instances of digital twins, which is no concern as long as they agree on their prediction of future events. However, when they yield different simulation results, there must exist a method of reconciliation between multiple digital twins. For this purpose, a level of trust may be assigned to each digital twin and subsequently computed based on the factual prediction quality of the simulation derived from a comparison with the observed events in the environment. By doing so, inapt digital twins may be sorted out, at least eventually. Treating simulations as just another form of knowledge sources, this approach mimicks the teacher/student pattern described in [9].

Finally, it is still to be researched how the digital twin can best be integrated into the development process and the overall system life cycle respectively. As the digital twin needs to accurately simulate every part of system hardware and system hardware may only be determined after testing multiple prototypes we are left with a classic chicken-and-egg problem. Intuitive processes that are able to develop a digital twin alongside a system specification while still disposing of a consistent system at least at most times during the development processes are most definitely sought for.

References

[1] E. A. Lee, "Cyber physical systems: Design challenges," in *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*. IEEE, 2008, pp. 363–369.

[2] R. R. Rajkumar, I. Lee, L. Sha, and J. Stankovic, "Cyber-physical systems: the next computing revolution," in *Proceedings of the 47th Design Automation Conference*. ACM, 2010, pp. 731–736.

[3] J. Morris, S. Zemerick, M. Grubb, J. Lucas, M. Jaridi, J. N. Gross, N. Ohi, J. A. Christian, D. Vassiliadis, A. Kadiyala *et al.*, "Simulation-to-flight (stf-1): A mission to enable cubesat software-based validation and verification," 2016.

[4] E. J. Tuegel, A. R. Ingraffea, T. G. Eason, and S. M. Spottswood, "Reengineering aircraft structural life prediction using a digital twin," *International Journal of Aerospace Engineering*, vol. 2011, 2011.

[5] E. H. Glaessgen and D. Stargel, "The digital twin paradigm for future nasa and us air force vehicles," in *53rd Struct. Dyn. Mater. Conf. Special Session: Digital Twin, Honolulu, HI, US*, 2012, pp. 1–14.

[6] A. Cerrone, J. Hochhalter, G. Heber, and A. Ingraffea, "On the effects of modeling as-manufactured geometry: Toward digital twin," *International Journal of Aerospace Engineering*, vol. 2014, 2014.

[7] L. Belzner, R. Hennicker, and M. Wirsing, "Onplan: A framework for simulation-based online planning," in *Formal Aspects of Component Software*. Springer, 2015, pp. 1–30.

[8] J. Vlissides, R. Helm, R. Johnson, and E. Gamma, "Design patterns: Elements of reusable object-oriented software," *Reading: Addison-Wesley*, vol. 49, no. 120, p. 11, 1995.

[9] M. Hölzl and T. Gabor, "Continuous collaboration: a case study on the development of an adaptive cyber-physical system," in *Software Engineering for Smart Cyber-Physical Systems (SEsCPS), 2015 IEEE/ACM 1st International Workshop on*. IEEE, 2015, pp. 19–25.

Taken from original publication: Thomas Gabor, Lenz Belzner, Marie Kiermeier, Michael Till Beck, and Alexander Neitz. A simulation-based architecture for smart cyber-physical systems. In *The International Workshop on Models@run.time for Self-Aware Computing Systems*, 2016

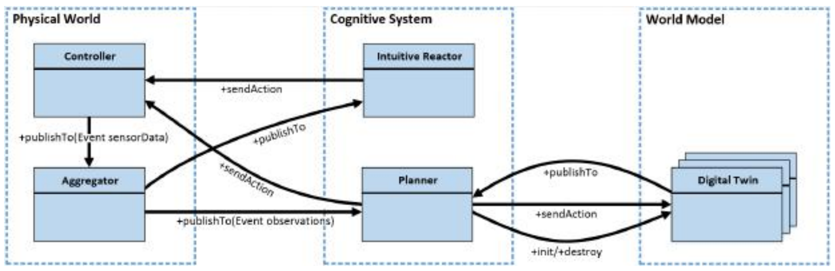


Fig. 3. Instance model of a software system using a digital twin. The dashed boxes mirror the setup shown in Figure 2.

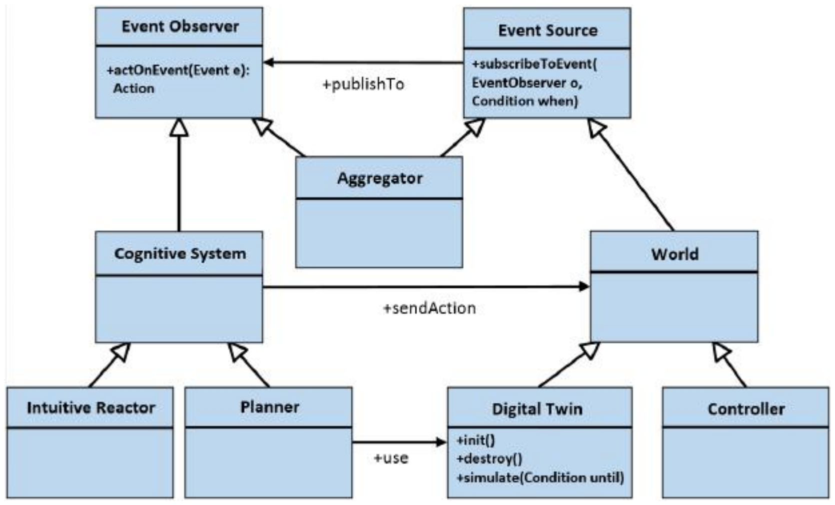
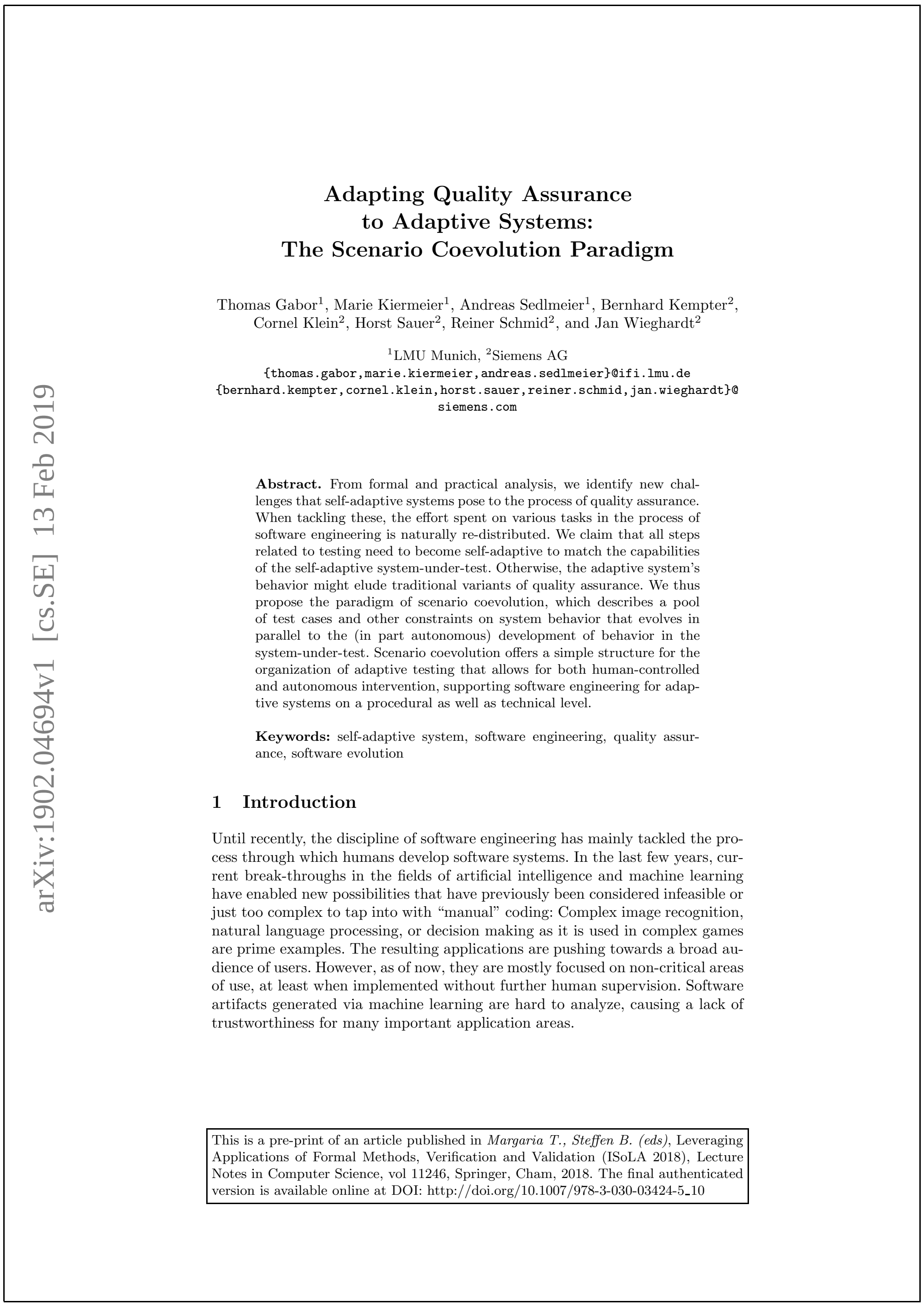


Fig. 4. Class diagram of a simulation-based architecture using a digital twin.

Taken from original publication: Thomas Gabor, Marie Kiermeier, Andreas Sedlmeier, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, and Jan Wieghardt. Adapting quality assurance to adaptive systems: The scenario coevolution paradigm. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2018



Adapting Quality Assurance to Adaptive Systems: The Scenario Coevolution Paradigm

Thomas Gabor[1], Marie Kiermeier[1], Andreas Sedlmeier[1], Bernhard Kempter[2], Cornel Klein[2], Horst Sauer[2], Reiner Schmid[2], and Jan Wieghardt[2]

[1]LMU Munich, [2]Siemens AG
{thomas.gabor,marie.kiermeier,andreas.sedlmeier}@ifi.lmu.de
{bernhard.kempter,cornel.klein,horst.sauer,reiner.schmid,jan.wieghardt}@siemens.com

Abstract. From formal and practical analysis, we identify new challenges that self-adaptive systems pose to the process of quality assurance. When tackling these, the effort spent on various tasks in the process of software engineering is naturally re-distributed. We claim that all steps related to testing need to become self-adaptive to match the capabilities of the self-adaptive system-under-test. Otherwise, the adaptive system's behavior might elude traditional variants of quality assurance. We thus propose the paradigm of scenario coevolution, which describes a pool of test cases and other constraints on system behavior that evolves in parallel to the (in part autonomous) development of behavior in the system-under-test. Scenario coevolution offers a simple structure for the organization of adaptive testing that allows for both human-controlled and autonomous intervention, supporting software engineering for adaptive systems on a procedural as well as technical level.

Keywords: self-adaptive system, software engineering, quality assurance, software evolution

1 Introduction

Until recently, the discipline of software engineering has mainly tackled the process through which humans develop software systems. In the last few years, current break-throughs in the fields of artificial intelligence and machine learning have enabled new possibilities that have previously been considered infeasible or just too complex to tap into with "manual" coding: Complex image recognition, natural language processing, or decision making as it is used in complex games are prime examples. The resulting applications are pushing towards a broad audience of users. However, as of now, they are mostly focused on non-critical areas of use, at least when implemented without further human supervision. Software artifacts generated via machine learning are hard to analyze, causing a lack of trustworthiness for many important application areas.

This is a pre-print of an article published in *Margaria T., Steffen B. (eds)*, Leveraging Applications of Formal Methods, Verification and Validation (ISoLA 2018), Lecture Notes in Computer Science, vol 11246, Springer, Cham, 2018. The final authenticated version is available online at DOI: http://doi.org/10.1007/978-3-030-03424-5_10

Taken from original publication: Thomas Gabor, Marie Kiermeier, Andreas Sedlmeier, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, and Jan Wieghardt. Adapting quality assurance to adaptive systems: The scenario coevolution paradigm. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2018

We claim that in order to reinstate levels of trustworthiness comparable to well-known classical approaches, we need not essentially reproduce the principles of classical software test but need to develop a new approach towards software testing. We suggest to develop a system and its test suite in a competitive setting where each sub-system tries to outwit the other. We call this approach *scenario coevolution* and attempt to show the necessity of such an approach. We hope that trust in that dynamic (or similar ones) can help to build a new process for quality assurance, even for hardly predictable systems.

Following a top-down approach to the issue, we start in Section 2 by introducing a formal framework for the description of systems. We augment it to also include the process of software and system development. Section 3 provides a short overview on related work. From literature review and practical experience, we introduce four core concepts for the engineering of adaptive systems in Section 4. In order to integrate these with our formal framework, Section 5 contains an introduction of our notion of scenarios and their application to an incremental software testing process. In Section 6 we discuss which effect scenario coevolution has on a selection of practical software engineering tasks and how it helps implement the core concepts. Finally, Section 7 provides a short conclusion.

2 Formal Framework

In this section we introduce a formal framework as a basis for our analysis. We first build upon the framework described in [1] to define adaptive systems and then proceed to reason about the influence of their inherent structure on software architecture.

2.1 Describing Adaptive Systems

We roughly adopt the formal definitions of our vocabulary related to the description of systems from [1]: We describe a system as an arbitrary relation over a set of variables.

Definition 1 (System [1])**.** Let I be a (finite or infinite) set, and let $\mathcal{V} = (V_i)_{i \in I}$ be a family of sets. A *system* of type $\mathcal{V}$ is a relation S of type $\mathcal{V}$.

Given a System S, an element $s \in S$ is called the state of the system. For practical purposes, we usually want to discern various parts of a system's state space. For this reason, parts of the system relation of type $\mathcal{V}$ given by an index set $J \subseteq I$, i.e., $(V_j)_{j \in J}$, may be considered *inputs* and other parts given by a different index set may be considered *outputs* [1]. Formally, this makes no difference to the system. Semantically, we usually compute the output parts of the system using the input parts.

We introduce two more designated sub-spaces of the system relation: *situation* and *behavior*. These notions correspond roughly to the intended meaning

Taken from original publication: Thomas Gabor, Marie Kiermeier, Andreas Sedlmeier, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, and Jan Wieghardt. Adapting quality assurance to adaptive systems: The scenario coevolution paradigm. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2018

of inputs and outputs mentioned before. The situation is the part of the system state space that fully encapsulates all information the system has about its state. This may include parts that the system does have full control over (which we would consider counter-intuitive when using the notion of "input"). The behavior encapsulates the parts of the system that can only be computed by applying the system relation. Likewise, this does *not* imply that the system has full control over the values. Furthermore, a system may have an *internal state*, which is parts of the state space that are neither included in the situation nor in the behavior. When we are not interested in the internal space, we can regard a system as a mapping from situations to behavior, written $S = X \overset{Z}{\rightsquigarrow} Y$ for situations X and behaviors Y, where Z is the internal state of the system S. Using these notions, we can more aptly define some properties on systems.

Further following the line of thought presented in [1], we want to build systems out of other systems. At the core of software engineering, there is the principle of re-use of components, which we want to mirror in our formalism.

Definition 2 (Composition)**.** Let S_1 and S_2 be systems of types $\mathcal{V}_1 = (V_{1,i})_{i \in I_1}$ and $\mathcal{V}_2 = (V_{2,i})_{i \in I_2}$, respectively. Let $\mathcal{R}(\mathcal{V})$ be the domain of all relations over $\mathcal{V}$. A *combination operator* $\otimes$ is a function such that $S_1 \otimes S_2 \in \mathcal{R}(\mathcal{V})$ for some family of sets $\mathcal{V}$ with $V_{1,1}, ..., V_{1,m}, V_{2,1}, ..., V_{2,n} \in \mathcal{V}$.[1] The application of a combination operator is called *composition*. The arguments to a combination operator are called *components*.

Composition is not only important to model software architecture within our formalism, but it also defines the formal framework for interaction: Two systems interact when they are combined using a combination operator $\otimes$ that ensures that the behavior of (at least) one system is recognized within the situation of (at least) another system.

Definition 3 (Interaction)**.** Let $S = S_1 \otimes S_2$ be a composition of type $\mathcal{V}$ of systems S_1 and S_2 of type $\mathcal{V}_1$ and $\mathcal{V}_2$, respectively, using a combination operator $\otimes$. If there exist a $V_1 \in \mathcal{V}_1$ and a $V_2 \in \mathcal{V}_2$ and a relation $R \in V_1 \times V_2$ so that for all states $s \in S$, $(proj(s, V_1), proj(s, V_2)) \in R$, then the components S_1 and S_2 interact with respect to R.

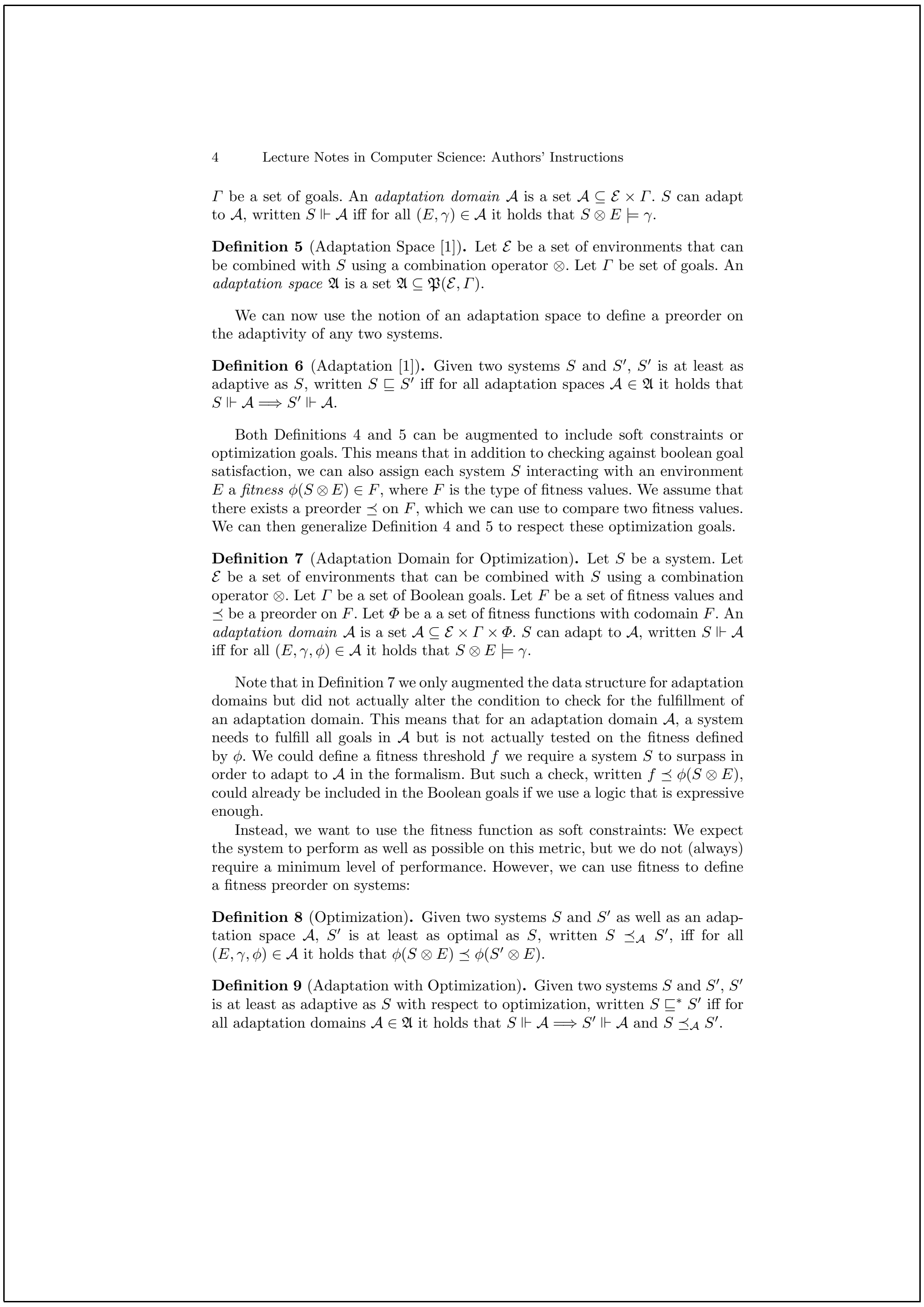
We can model an open system S as a combination $S = C \otimes E$ of a core system C and its environment E, both being modeled as systems again.

Hiding some of the complexity described in [1], we assume we have a logic $\mathfrak{L}$ in which we can express a system goal γ. We can always decide if γ holds for a given system, in which case we write $S \models \gamma$ for $\gamma(S) = \top$. Based on [1], we can use this concept to define an adaptation domain:

Definition 4 (Adaptation Domain [1])**.** Let S be a system. Let $\mathcal{E}$ be a set of environments that can be combined with S using a combination operator $\otimes$. Let

[1] In [1], there is a more strict definition on how the combination operator needs to handle the designated inputs and outputs of its given systems. Here, we opt for a more general definition.

Taken from original publication: Thomas Gabor, Marie Kiermeier, Andreas Sedlmeier, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, and Jan Wieghardt. Adapting quality assurance to adaptive systems: The scenario coevolution paradigm. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2018



Γ be a set of goals. An *adaptation domain* $\mathcal{A}$ is a set $\mathcal{A} \subseteq \mathcal{E} \times \Gamma$. S can adapt to $\mathcal{A}$, written $S \Vdash \mathcal{A}$ iff for all $(E, \gamma) \in \mathcal{A}$ it holds that $S \otimes E \models \gamma$.

Definition 5 (Adaptation Space [1])**.** Let $\mathcal{E}$ be a set of environments that can be combined with S using a combination operator $\otimes$. Let Γ be set of goals. An *adaptation space* $\mathfrak{A}$ is a set $\mathfrak{A} \subseteq \mathfrak{P}(\mathcal{E}, \Gamma)$.

We can now use the notion of an adaptation space to define a preorder on the adaptivity of any two systems.

Definition 6 (Adaptation [1])**.** Given two systems S and S', S' is at least as adaptive as S, written $S \sqsubseteq S'$ iff for all adaptation spaces $\mathcal{A} \in \mathfrak{A}$ it holds that $S \Vdash \mathcal{A} \Longrightarrow S' \Vdash \mathcal{A}$.

Both Definitions 4 and 5 can be augmented to include soft constraints or optimization goals. This means that in addition to checking against boolean goal satisfaction, we can also assign each system S interacting with an environment E a *fitness* $\phi(S \otimes E) \in F$, where F is the type of fitness values. We assume that there exists a preorder $\preceq$ on F, which we can use to compare two fitness values. We can then generalize Definition 4 and 5 to respect these optimization goals.

Definition 7 (Adaptation Domain for Optimization)**.** Let S be a system. Let $\mathcal{E}$ be a set of environments that can be combined with S using a combination operator $\otimes$. Let Γ be a set of Boolean goals. Let F be a set of fitness values and $\preceq$ be a preorder on F. Let Φ be a a set of fitness functions with codomain F. An *adaptation domain* $\mathcal{A}$ is a set $\mathcal{A} \subseteq \mathcal{E} \times \Gamma \times \Phi$. S can adapt to $\mathcal{A}$, written $S \Vdash \mathcal{A}$ iff for all $(E, \gamma, \phi) \in \mathcal{A}$ it holds that $S \otimes E \models \gamma$.

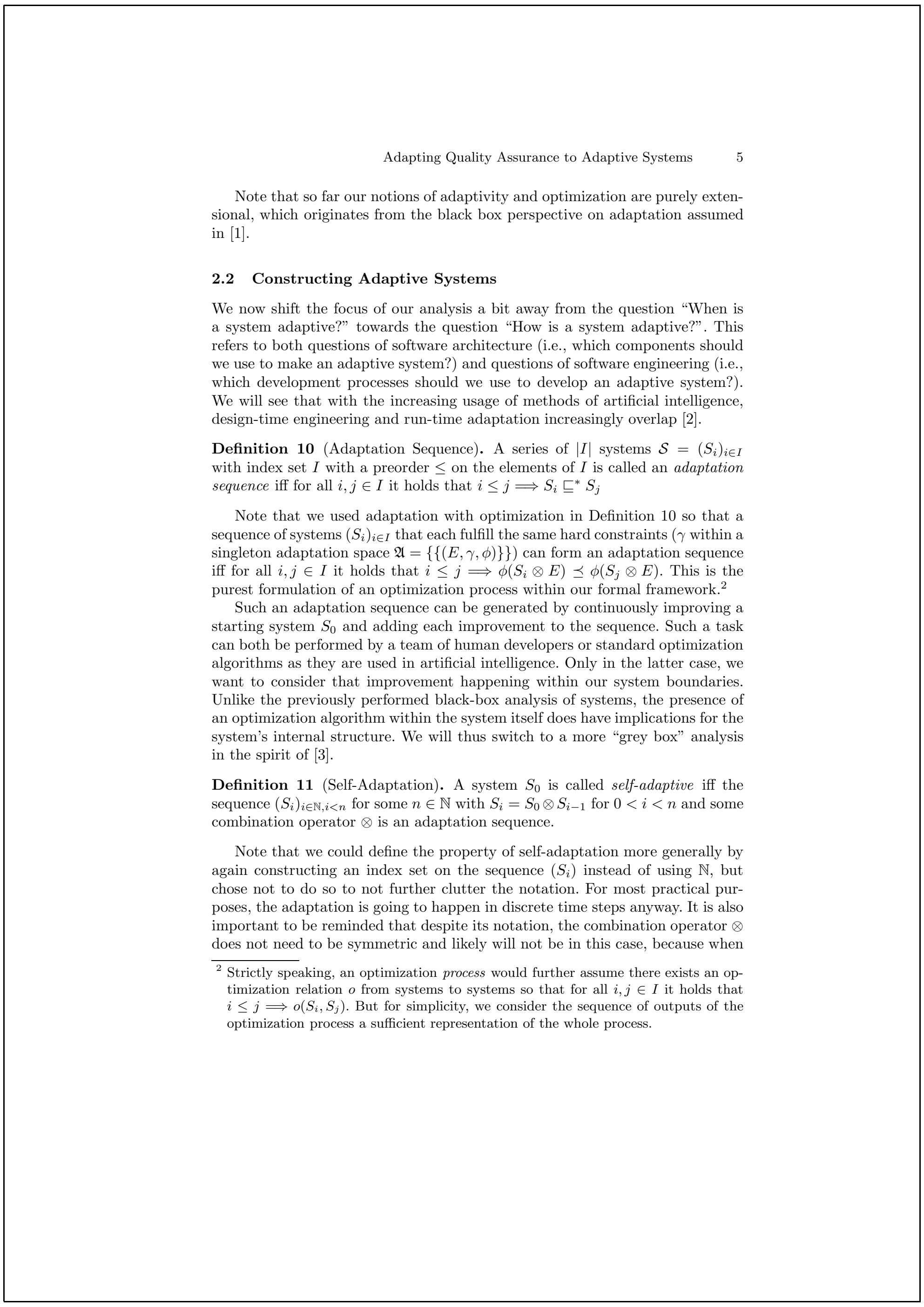
Note that in Definition 7 we only augmented the data structure for adaptation domains but did not actually alter the condition to check for the fulfillment of an adaptation domain. This means that for an adaptation domain $\mathcal{A}$, a system needs to fulfill all goals in $\mathcal{A}$ but is not actually tested on the fitness defined by ϕ. We could define a fitness threshold f we require a system S to surpass in order to adapt to $\mathcal{A}$ in the formalism. But such a check, written $f \preceq \phi(S \otimes E)$, could already be included in the Boolean goals if we use a logic that is expressive enough.

Instead, we want to use the fitness function as soft constraints: We expect the system to perform as well as possible on this metric, but we do not (always) require a minimum level of performance. However, we can use fitness to define a fitness preorder on systems:

Definition 8 (Optimization)**.** Given two systems S and S' as well as an adaptation space $\mathcal{A}$, S' is at least as optimal as S, written $S \preceq_{\mathcal{A}} S'$, iff for all $(E, \gamma, \phi) \in \mathcal{A}$ it holds that $\phi(S \otimes E) \preceq \phi(S' \otimes E)$.

Definition 9 (Adaptation with Optimization)**.** Given two systems S and S', S' is at least as adaptive as S with respect to optimization, written $S \sqsubseteq^* S'$ iff for all adaptation domains $\mathcal{A} \in \mathfrak{A}$ it holds that $S \Vdash \mathcal{A} \Longrightarrow S' \Vdash \mathcal{A}$ and $S \preceq_{\mathcal{A}} S'$.

Taken from original publication: Thomas Gabor, Marie Kiermeier, Andreas Sedlmeier, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, and Jan Wieghardt. Adapting quality assurance to adaptive systems: The scenario coevolution paradigm. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2018



Note that so far our notions of adaptivity and optimization are purely extensional, which originates from the black box perspective on adaptation assumed in [1].

2.2 Constructing Adaptive Systems

We now shift the focus of our analysis a bit away from the question "When is a system adaptive?" towards the question "How is a system adaptive?". This refers to both questions of software architecture (i.e., which components should we use to make an adaptive system?) and questions of software engineering (i.e., which development processes should we use to develop an adaptive system?). We will see that with the increasing usage of methods of artificial intelligence, design-time engineering and run-time adaptation increasingly overlap [2].

Definition 10 (Adaptation Sequence)**.** A series of $|I|$ systems $\mathcal{S} = (S_i)_{i \in I}$ with index set I with a preorder $\leq$ on the elements of I is called an *adaptation sequence* iff for all $i, j \in I$ it holds that $i \leq j \Longrightarrow S_i \sqsubseteq^* S_j$

Note that we used adaptation with optimization in Definition 10 so that a sequence of systems $(S_i)_{i \in I}$ that each fulfill the same hard constraints (γ within a singleton adaptation space $\mathfrak{A} = \{\{(E, \gamma, \phi)\}\}$) can form an adaptation sequence iff for all $i, j \in I$ it holds that $i \leq j \Longrightarrow \phi(S_i \otimes E) \preceq \phi(S_j \otimes E)$. This is the purest formulation of an optimization process within our formal framework.[2]

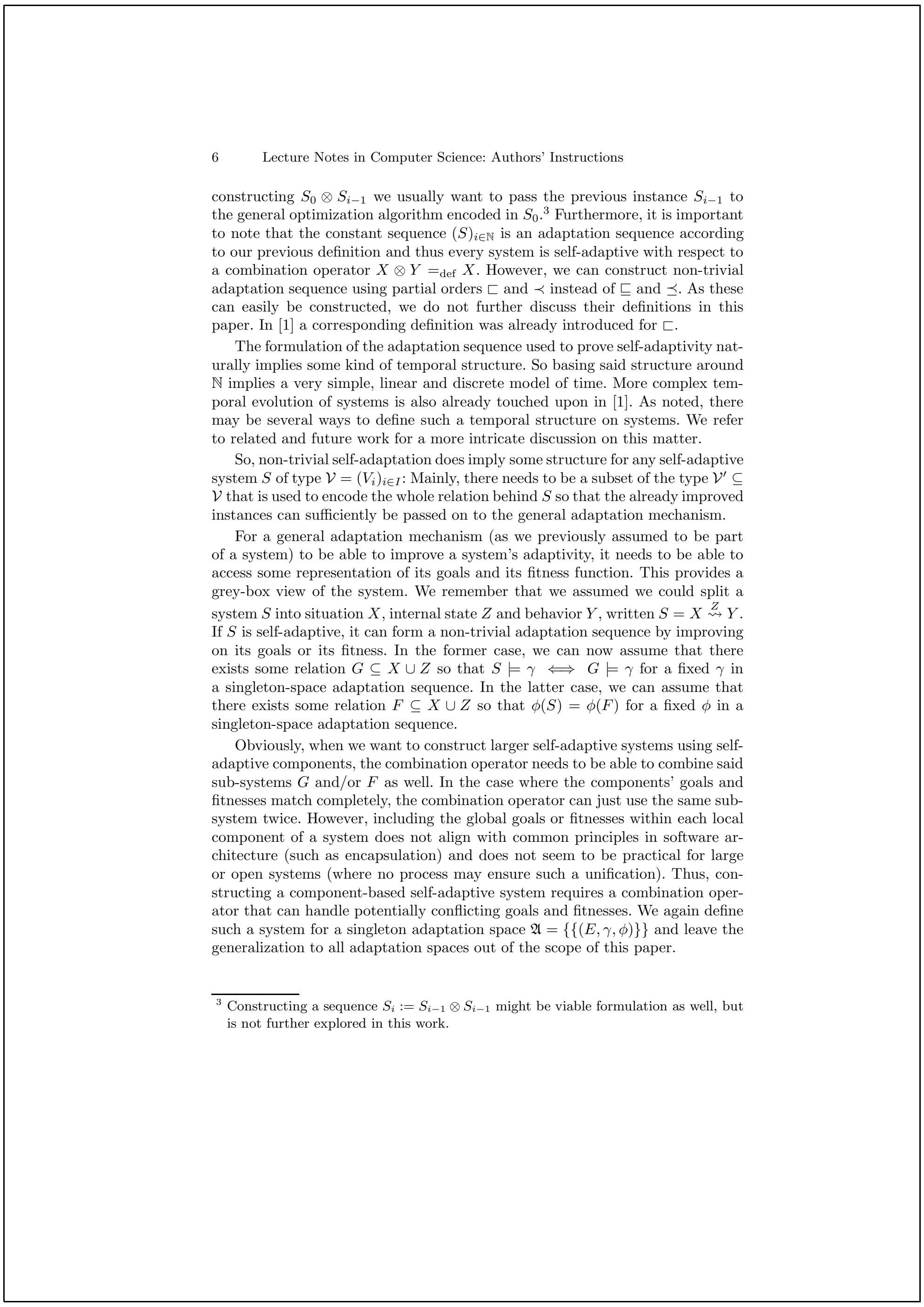
Such an adaptation sequence can be generated by continuously improving a starting system S_0 and adding each improvement to the sequence. Such a task can both be performed by a team of human developers or standard optimization algorithms as they are used in artificial intelligence. Only in the latter case, we want to consider that improvement happening within our system boundaries. Unlike the previously performed black-box analysis of systems, the presence of an optimization algorithm within the system itself does have implications for the system's internal structure. We will thus switch to a more "grey box" analysis in the spirit of [3].

Definition 11 (Self-Adaptation)**.** A system S_0 is called *self-adaptive* iff the sequence $(S_i)_{i \in \mathbb{N}, i < n}$ for some $n \in \mathbb{N}$ with $S_i = S_0 \otimes S_{i-1}$ for $0 < i < n$ and some combination operator $\otimes$ is an adaptation sequence.

Note that we could define the property of self-adaptation more generally by again constructing an index set on the sequence (S_i) instead of using $\mathbb{N}$, but chose not to do so to not further clutter the notation. For most practical purposes, the adaptation is going to happen in discrete time steps anyway. It is also important to be reminded that despite its notation, the combination operator $\otimes$ does not need to be symmetric and likely will not be in this case, because when

[2] Strictly speaking, an optimization *process* would further assume there exists an optimization relation o from systems to systems so that for all $i, j \in I$ it holds that $i \leq j \Longrightarrow o(S_i, S_j)$. But for simplicity, we consider the sequence of outputs of the optimization process a sufficient representation of the whole process.

Taken from original publication: Thomas Gabor, Marie Kiermeier, Andreas Sedlmeier, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, and Jan Wieghardt. Adapting quality assurance to adaptive systems: The scenario coevolution paradigm. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2018



constructing $S_0 \otimes S_{i-1}$ we usually want to pass the previous instance S_{i-1} to the general optimization algorithm encoded in S_0.[3] Furthermore, it is important to note that the constant sequence $(S)_{i \in \mathbb{N}}$ is an adaptation sequence according to our previous definition and thus every system is self-adaptive with respect to a combination operator $X \otimes Y =_{\text{def}} X$. However, we can construct non-trivial adaptation sequence using partial orders $\sqsubset$ and $\prec$ instead of $\sqsubseteq$ and $\preceq$. As these can easily be constructed, we do not further discuss their definitions in this paper. In [1] a corresponding definition was already introduced for $\sqsubset$.

The formulation of the adaptation sequence used to prove self-adaptivity naturally implies some kind of temporal structure. So basing said structure around $\mathbb{N}$ implies a very simple, linear and discrete model of time. More complex temporal evolution of systems is also already touched upon in [1]. As noted, there may be several ways to define such a temporal structure on systems. We refer to related and future work for a more intricate discussion on this matter.

So, non-trivial self-adaptation does imply some structure for any self-adaptive system S of type $\mathcal{V} = (V_i)_{i \in I}$: Mainly, there needs to be a subset of the type $\mathcal{V}' \subseteq \mathcal{V}$ that is used to encode the whole relation behind S so that the already improved instances can sufficiently be passed on to the general adaptation mechanism.

For a general adaptation mechanism (as we previously assumed to be part of a system) to be able to improve a system's adaptivity, it needs to be able to access some representation of its goals and its fitness function. This provides a grey-box view of the system. We remember that we assumed we could split a system S into situation X, internal state Z and behavior Y, written $S = X \overset{Z}{\rightsquigarrow} Y$. If S is self-adaptive, it can form a non-trivial adaptation sequence by improving on its goals or its fitness. In the former case, we can now assume that there exists some relation $G \subseteq X \cup Z$ so that $S \models \gamma \iff G \models \gamma$ for a fixed γ in a singleton-space adaptation sequence. In the latter case, we can assume that there exists some relation $F \subseteq X \cup Z$ so that $\phi(S) = \phi(F)$ for a fixed ϕ in a singleton-space adaptation sequence.

Obviously, when we want to construct larger self-adaptive systems using self-adaptive components, the combination operator needs to be able to combine said sub-systems G and/or F as well. In the case where the components' goals and fitnesses match completely, the combination operator can just use the same sub-system twice. However, including the global goals or fitnesses within each local component of a system does not align with common principles in software architecture (such as encapsulation) and does not seem to be practical for large or open systems (where no process may ensure such a unification). Thus, constructing a component-based self-adaptive system requires a combination operator that can handle potentially conflicting goals and fitnesses. We again define such a system for a singleton adaptation space $\mathfrak{A} = \{\{(E, \gamma, \phi)\}\}$ and leave the generalization to all adaptation spaces out of the scope of this paper.

[3] Constructing a sequence $S_i := S_{i-1} \otimes S_{i-1}$ might be viable formulation as well, but is not further explored in this work.

Taken from original publication: Thomas Gabor, Marie Kiermeier, Andreas Sedlmeier, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, and Jan Wieghardt. Adapting quality assurance to adaptive systems: The scenario coevolution paradigm. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2018

Definition 12 (Multi-Agent System)**.** Given a system $S = S_1 \otimes ... \otimes S_n$ that adapts to $\mathcal{A} = \{(E, \gamma, \phi)\}$. Iff for each $1 \leq i \leq n$ with $i, n \in \mathbb{N}, n > 1$ there is an adaptation domain $\mathcal{A}_i = \{(E_i, \gamma_i, \phi_i)\}$ so that (1) $E_i = E \otimes S_1 \otimes ... \otimes S_{i-1} \otimes S_{i+1} \otimes ... \otimes S_n$ and (2) $\gamma_i \neq \gamma$ or $\phi_i \neq \phi$ and (3) S_i adapts to $\mathcal{A}_i$, then S is a *multi-agent system* with agents $S_1, ..., S_n$.

For practical purposes, we usually want to use the notion of multi-agent systems in a transistive way, i.e., we can call a system a multi-agent system as soon as any part of it is a multi-agent system according to Definition12. Formally, S is a multi-agent system if there are systems components S', R so that $S = S' \otimes R$ and S' is a multi-agent system. We argue that this transitivity is not only justified but a crucial point for systems development of adaptive systems: Agents tend to utilize their environment to fulfill their own goals and can thus "leak" their goals into other system components. Not that Condition (2) of Definition 12 ensures that not every system constructed by composition is regarded a multi-agent system; it is necessary to feature agents with (at least slightly) differing adaptation properties.

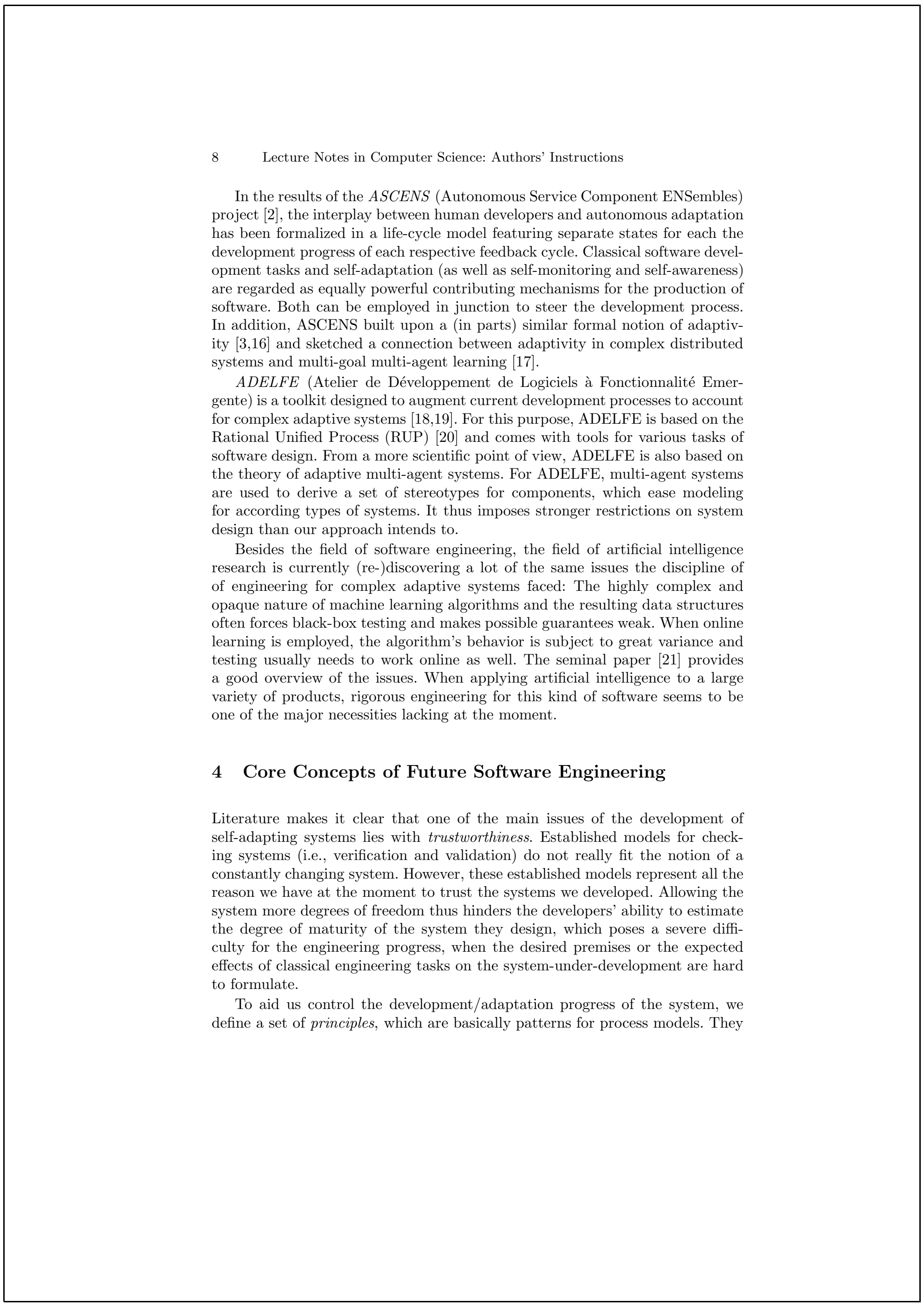
For the remainder of this paper, we will apply Definition 12 "backwards": Whenever we look at a self-adaptive system S, whose goals or fitnesses can be split into several sub-goals or sub-fitnesses we can regard S as a multi-agent system. Using this knowledge, we can apply design patterns from multi-agent systems to all self-adaptive systems without loss of generality. Furthermore, we need to be aware that especially if we do not explicitly design multi-agent coordination between different sub-goals, such a coordination will be done implicitly. Essentially, there is no way around generalizing software engineering approaches for self-adaptive systems to potentially adversarial components.

3 Related Work

Many researchers and practitioners in recent years have already been concerned about the changes necessary to allow for solid and reliable software engineering processes for (self-)adaptive systems. Central challenges were collected in [4], where issues of quality assurance are already mentioned but the focus is more on bringing about complex adaptive behavior in the first place. The later research roadmap of [5] puts a strong focus on interaction patterns of already adaptive systems (both between each other and with human developers) and already dedicates a section to verification and validation issues, being close in mind to the perspective of this work. We fall in line with the roadmap further specified in [6,7,8].

While this work largely builds upon [1], there have been other approaches to formalize the notion of adaptivity: [9] discusses high-level architectural patterns that form multiple inter-connected adaptation loops. In [10] such feedback loops are based on the MAPE-K model [11]. While these approaches largely focus on the formal construction of adaptive systems, there have also been approaches that assume a (more human-centric or at least tool-centric) software engineering perspective [12,13,14,15]. We want to discuss two of those on greater detail:

Taken from original publication: Thomas Gabor, Marie Kiermeier, Andreas Sedlmeier, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, and Jan Wieghardt. Adapting quality assurance to adaptive systems: The scenario coevolution paradigm. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2018



In the results of the *ASCENS* (Autonomous Service Component ENSembles) project [2], the interplay between human developers and autonomous adaptation has been formalized in a life-cycle model featuring separate states for each the development progress of each respective feedback cycle. Classical software development tasks and self-adaptation (as well as self-monitoring and self-awareness) are regarded as equally powerful contributing mechanisms for the production of software. Both can be employed in junction to steer the development process. In addition, ASCENS built upon a (in parts) similar formal notion of adaptivity [3,16] and sketched a connection between adaptivity in complex distributed systems and multi-goal multi-agent learning [17].

ADELFE (Atelier de Développement de Logiciels à Fonctionnalité Emergente) is a toolkit designed to augment current development processes to account for complex adaptive systems [18,19]. For this purpose, ADELFE is based on the Rational Unified Process (RUP) [20] and comes with tools for various tasks of software design. From a more scientific point of view, ADELFE is also based on the theory of adaptive multi-agent systems. For ADELFE, multi-agent systems are used to derive a set of stereotypes for components, which ease modeling for according types of systems. It thus imposes stronger restrictions on system design than our approach intends to.

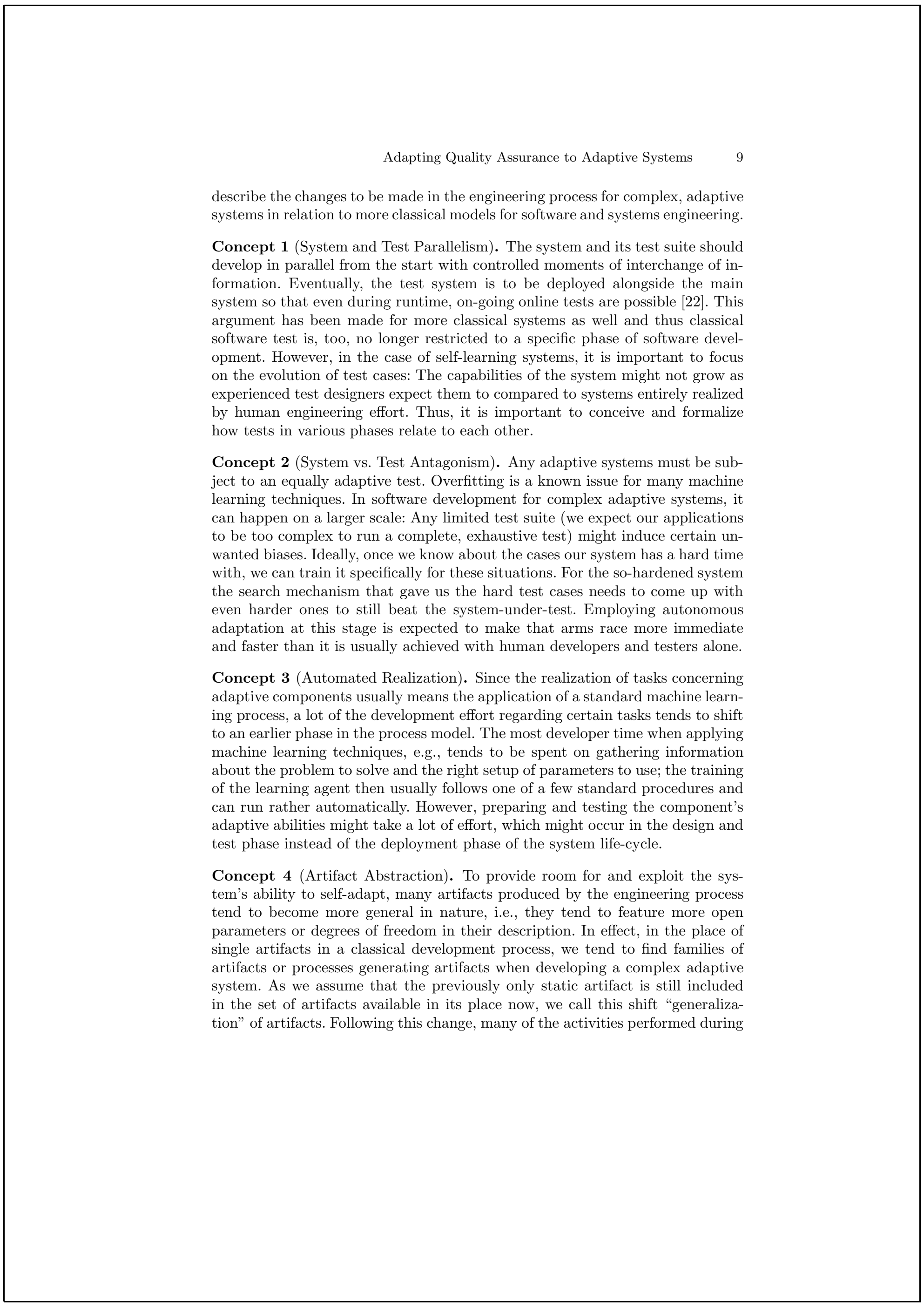
Besides the field of software engineering, the field of artificial intelligence research is currently (re-)discovering a lot of the same issues the discipline of of engineering for complex adaptive systems faced: The highly complex and opaque nature of machine learning algorithms and the resulting data structures often forces black-box testing and makes possible guarantees weak. When online learning is employed, the algorithm's behavior is subject to great variance and testing usually needs to work online as well. The seminal paper [21] provides a good overview of the issues. When applying artificial intelligence to a large variety of products, rigorous engineering for this kind of software seems to be one of the major necessities lacking at the moment.

4 Core Concepts of Future Software Engineering

Literature makes it clear that one of the main issues of the development of self-adapting systems lies with *trustworthiness*. Established models for checking systems (i.e., verification and validation) do not really fit the notion of a constantly changing system. However, these established models represent all the reason we have at the moment to trust the systems we developed. Allowing the system more degrees of freedom thus hinders the developers' ability to estimate the degree of maturity of the system they design, which poses a severe difficulty for the engineering progress, when the desired premises or the expected effects of classical engineering tasks on the system-under-development are hard to formulate.

To aid us control the development/adaptation progress of the system, we define a set of *principles*, which are basically patterns for process models. They

Taken from original publication: Thomas Gabor, Marie Kiermeier, Andreas Sedlmeier, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, and Jan Wieghardt. Adapting quality assurance to adaptive systems: The scenario coevolution paradigm. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2018



describe the changes to be made in the engineering process for complex, adaptive systems in relation to more classical models for software and systems engineering.

Concept 1 (System and Test Parallelism)**.** The system and its test suite should develop in parallel from the start with controlled moments of interchange of information. Eventually, the test system is to be deployed alongside the main system so that even during runtime, on-going online tests are possible [22]. This argument has been made for more classical systems as well and thus classical software test is, too, no longer restricted to a specific phase of software development. However, in the case of self-learning systems, it is important to focus on the evolution of test cases: The capabilities of the system might not grow as experienced test designers expect them to compared to systems entirely realized by human engineering effort. Thus, it is important to conceive and formalize how tests in various phases relate to each other.

Concept 2 (System vs. Test Antagonism)**.** Any adaptive systems must be subject to an equally adaptive test. Overfitting is a known issue for many machine learning techniques. In software development for complex adaptive systems, it can happen on a larger scale: Any limited test suite (we expect our applications to be too complex to run a complete, exhaustive test) might induce certain unwanted biases. Ideally, once we know about the cases our system has a hard time with, we can train it specifically for these situations. For the so-hardened system the search mechanism that gave us the hard test cases needs to come up with even harder ones to still beat the system-under-test. Employing autonomous adaptation at this stage is expected to make that arms race more immediate and faster than it is usually achieved with human developers and testers alone.

Concept 3 (Automated Realization)**.** Since the realization of tasks concerning adaptive components usually means the application of a standard machine learning process, a lot of the development effort regarding certain tasks tends to shift to an earlier phase in the process model. The most developer time when applying machine learning techniques, e.g., tends to be spent on gathering information about the problem to solve and the right setup of parameters to use; the training of the learning agent then usually follows one of a few standard procedures and can run rather automatically. However, preparing and testing the component's adaptive abilities might take a lot of effort, which might occur in the design and test phase instead of the deployment phase of the system life-cycle.

Concept 4 (Artifact Abstraction)**.** To provide room for and exploit the system's ability to self-adapt, many artifacts produced by the engineering process tend to become more general in nature, i.e., they tend to feature more open parameters or degrees of freedom in their description. In effect, in the place of single artifacts in a classical development process, we tend to find families of artifacts or processes generating artifacts when developing a complex adaptive system. As we assume that the previously only static artifact is still included in the set of artifacts available in its place now, we call this shift "generalization" of artifacts. Following this change, many of the activities performed during

Taken from original publication: Thomas Gabor, Marie Kiermeier, Andreas Sedlmeier, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, and Jan Wieghardt. Adapting quality assurance to adaptive systems: The scenario coevolution paradigm. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2018

development shift their targets from concrete implementations to more general artifact, i.e., when building a test suite no longer yields a series of runnable test cases but instead produces a test case generator. When this principle is broadly applied, the development activities shift towards "meta development". The developers are concerned with setting up a process able to find good solutions autonomously instead of finding the good solutions directly.

5 Scenarios

We now want to include the issue of testing adaptive systems in our formal framework. We recognize that any development process for systems following the principles described in Section 2 produces two central types of artifacts: The first one is a system $S = X \overset{Z}{\rightsquigarrow} Y$ with a specific desired behavior Y so that it manages to adapt to a given adaptation space. The second is a set of situations, test cases, constraints, and checked properties that this system's behavior has been validated against. We call artifacts of the second type by the group name of *scenarios*.

Definition 13 (Scenario)**.** Let $S = X \overset{Z}{\rightsquigarrow} Y$ be a system and $\mathcal{A} = \{(E, \gamma, \phi)\}$ a singleton adaptation domain. A tuple $c = (X, Y, g, f), g \in \{\top, \bot\}, f \in \text{cod}(\phi)$ with $g = \top \iff S \otimes E \models \gamma$ and $f = \phi(S \otimes E)$ is called *scenario*.[4]

Semantically, scenarios represent the experience gained about the system's behavior during development, including both successful ($S \models \gamma$) and unsuccessful ($S \not\models \gamma$) test runs. As stated above, since we expect to operate in test spaces we cannot cover exhaustively, the knowledge about the areas we did cover is an important asset and likewise result of the systems engineering process.

Effectively, as we construct and evolve a system S we want to construct and augment a set of scenarios $C = \{c_1, ..., c_n\}$ alongside with it. C is also called a *scenario suite* and can be seen as a toolbox to test S's adaptation abilities with respect to a fixed adaptation domain $\mathcal{A}$.

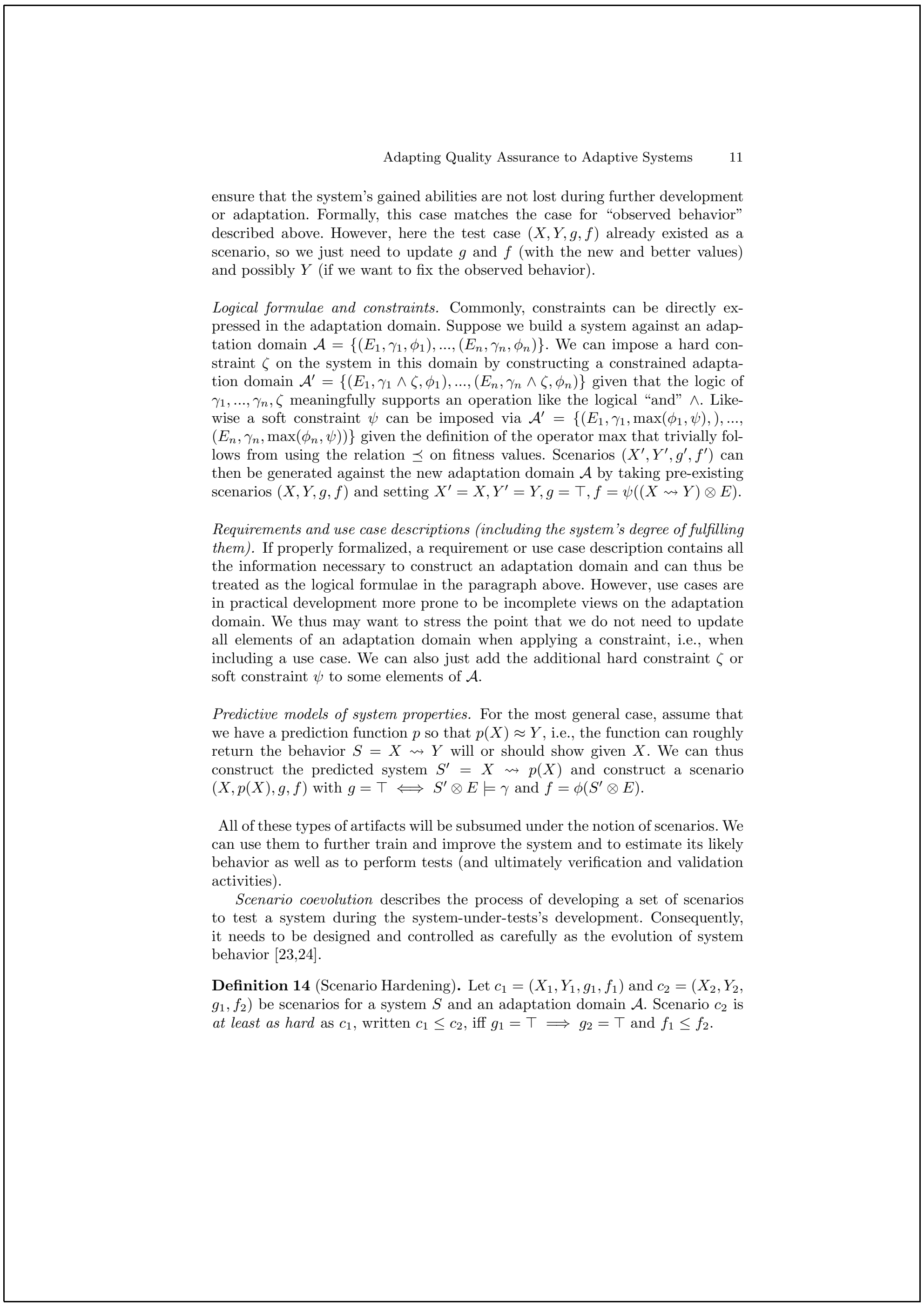
While formally abiding to Definition 13, scenarios can be encoded in various ways in practical software development, such as:

Sets of data points of expected or observed behavior. Given a system $S' = X' \rightsquigarrow Y'$ whose behavior is desirable (for example a trained predecessor of our system or a watchdog component), we can create scenarios (X', Y', g', f') with $g' = \top \iff S' \otimes E_i \models \gamma_i$ and $f' = \phi_i(S' \otimes E_i)$ for an arbitrary amount of elements (E_i, γ_i, ϕ_i) of an adaptation domain $\mathcal{A} = \{(E_1, \gamma_1, \phi_1), ..., (E_n, \gamma_n, \phi_n)\}$.

Test cases the system mastered. In some cases, adaptive systems may produce innovative behavior before we actively seek it out. In this cases, it is helpful to formalize the produced results once they have been found so that we can

[4] If we are only interested in the system's performance and not *how* it was achieved, we can redefine a scenario to leave out Y.

Taken from original publication: Thomas Gabor, Marie Kiermeier, Andreas Sedlmeier, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, and Jan Wieghardt. Adapting quality assurance to adaptive systems: The scenario coevolution paradigm. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2018



ensure that the system's gained abilities are not lost during further development or adaptation. Formally, this case matches the case for "observed behavior" described above. However, here the test case (X, Y, g, f) already existed as a scenario, so we just need to update g and f (with the new and better values) and possibly Y (if we want to fix the observed behavior).

Logical formulae and constraints. Commonly, constraints can be directly expressed in the adaptation domain. Suppose we build a system against an adaptation domain $\mathcal{A} = \{(E_1, \gamma_1, \phi_1), ..., (E_n, \gamma_n, \phi_n)\}$. We can impose a hard constraint ζ on the system in this domain by constructing a constrained adaptation domain $\mathcal{A}' = \{(E_1, \gamma_1 \wedge \zeta, \phi_1), ..., (E_n, \gamma_n \wedge \zeta, \phi_n)\}$ given that the logic of $\gamma_1, ..., \gamma_n, \zeta$ meaningfully supports an operation like the logical "and" $\wedge$. Likewise a soft constraint ψ can be imposed via $\mathcal{A}' = \{(E_1, \gamma_1, \max(\phi_1, \psi),), ..., (E_n, \gamma_n, \max(\phi_n, \psi))\}$ given the definition of the operator max that trivially follows from using the relation $\preceq$ on fitness values. Scenarios (X', Y', g', f') can then be generated against the new adaptation domain $\mathcal{A}$ by taking pre-existing scenarios (X, Y, g, f) and setting $X' = X, Y' = Y, g = \top, f = \psi((X \rightsquigarrow Y) \otimes E)$.

Requirements and use case descriptions (including the system's degree of fulfilling them). If properly formalized, a requirement or use case description contains all the information necessary to construct an adaptation domain and can thus be treated as the logical formulae in the paragraph above. However, use cases are in practical development more prone to be incomplete views on the adaptation domain. We thus may want to stress the point that we do not need to update all elements of an adaptation domain when applying a constraint, i.e., when including a use case. We can also just add the additional hard constraint ζ or soft constraint ψ to some elements of $\mathcal{A}$.

Predictive models of system properties. For the most general case, assume that we have a prediction function p so that $p(X) \approx Y$, i.e., the function can roughly return the behavior $S = X \rightsquigarrow Y$ will or should show given X. We can thus construct the predicted system $S' = X \rightsquigarrow p(X)$ and construct a scenario $(X, p(X), g, f)$ with $g = \top \iff S' \otimes E \models \gamma$ and $f = \phi(S' \otimes E)$.

All of these types of artifacts will be subsumed under the notion of scenarios. We can use them to further train and improve the system and to estimate its likely behavior as well as to perform tests (and ultimately verification and validation activities).

Scenario coevolution describes the process of developing a set of scenarios to test a system during the system-under-tests's development. Consequently, it needs to be designed and controlled as carefully as the evolution of system behavior [23,24].

Definition 14 (Scenario Hardening). Let $c_1 = (X_1, Y_1, g_1, f_1)$ and $c_2 = (X_2, Y_2, g_1, f_2)$ be scenarios for a system S and an adaptation domain $\mathcal{A}$. Scenario c_2 is *at least as hard* as c_1, written $c_1 \leq c_2$, iff $g_1 = \top \implies g_2 = \top$ and $f_1 \leq f_2$.

Taken from original publication: Thomas Gabor, Marie Kiermeier, Andreas Sedlmeier, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, and Jan Wieghardt. Adapting quality assurance to adaptive systems: The scenario coevolution paradigm. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2018

Definition 15 (Scenario Suite Order)**.** Let $C = \{c_1, ..., c_m\}$ and $C' = \{c'_1, ..., c'_n\}$ be sets of scenarios, also called scenarios suites. Scenario suite C' is *at least as hard* as C, written $C \sqsubseteq C'$, iff for all scenarios $c \in C$ there exists a scenario $c' \in C'$ so that $c \leq c'$.

Definition 16 (Scenario Sequence)**.** Let $\mathcal{S} = (S_i)_{i \in I}, I = \{1, ..., n\}$ be an adaptation sequence for a singleton adaptation space $\mathfrak{A} = \{\mathcal{A}\}$. A series of sets $\mathcal{C} = (C_i)_{i \in I}$ is called a scenario sequence iff for all $i \in I, i < n$ it holds that C_i is a scenario suite for S_i and $\mathcal{A}$ and $C_i \sqsubseteq C_{i+1}$.

We expect each phase of development to further alter the set of scenarios just as it does alter the system behavior. The scenarios produced and used at a certain phase in development must match the current state of progress. Valid scenarios from previous phases should be kept and checked against the further specialized system. When we do not delete any scenarios entirely, the continued addition of scenarios will ideally narrow down allowed system behavior to the desired possibilities. Eventually, we expect all activities of system test to be expressible as the generation or evaluation of scenarios. New scenarios may simply be thought up by system developers or be generated automatically.

Finding the right scenarios to generate is another optimization problem to be solved during the development of any complex adaptive system. Scenario evolution represents a cross-cutting concern for all phases of system development. Treating scenarios as first-class citizen among the artifacts produced by system development thus yields changes in tasks throughout the whole process model.

6 Applications of Scenario Coevolution

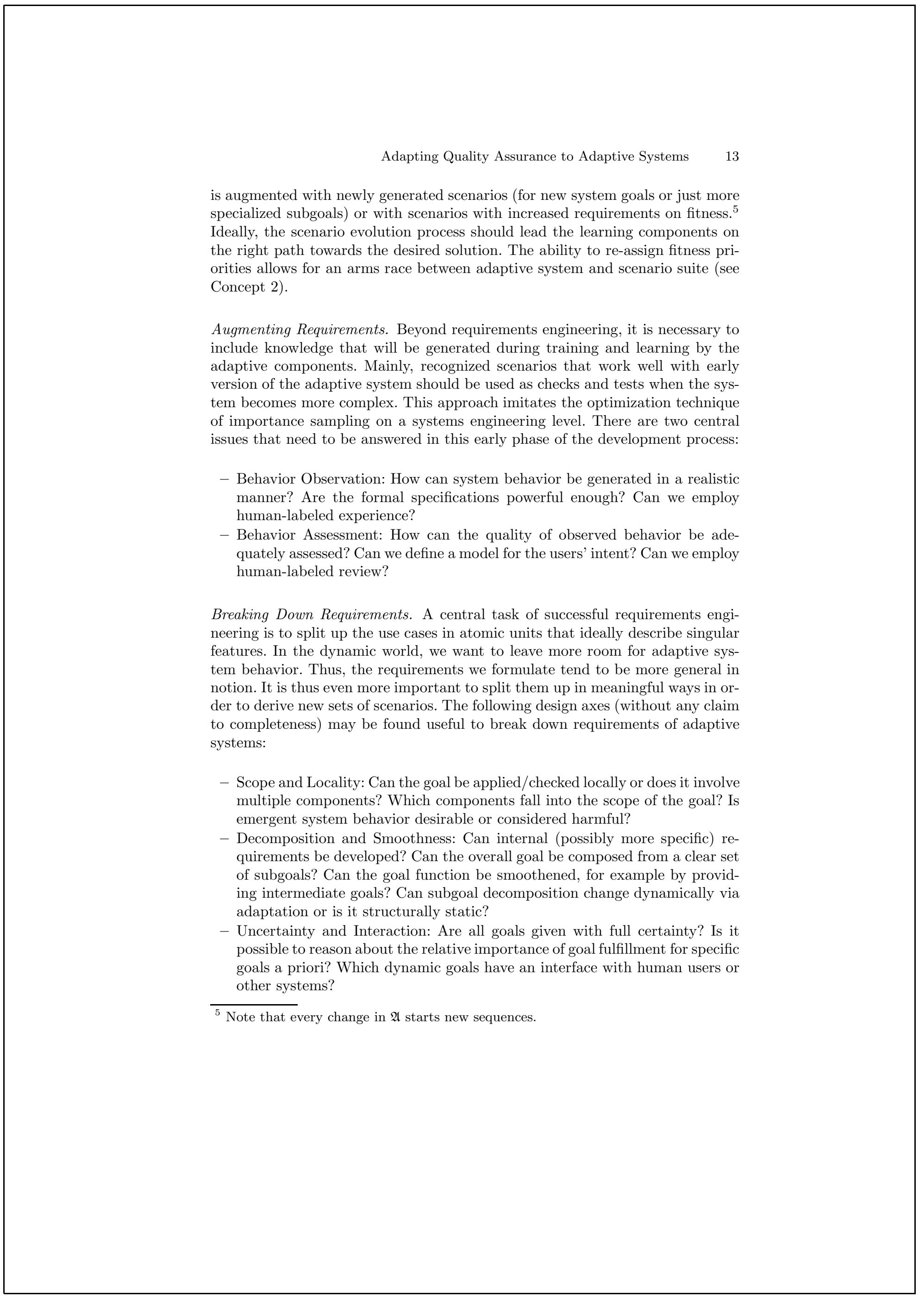
Having both introduced a formal framework for adaptation and the testing of adaptive systems using scenarios, we show in this section how these frameworks can be applied to aid the trustworthiness of complex adaptive systems for practical use.

6.1 Criticality Focus

It is very important to start the scenario evolution process alongside the system evolution, so that at each stage there exists a set of scenarios available to test the system's functionality and degree of progress (see Concept 1). This approach mimics the concept of agile development where between each sprint there exists a fully functional (however incomplete) version of the system. The ceoncept of scenario evolution integrates seamlessly with agile process models.

In the early phases of development, the common artifacts of requirements engineering, i.e., formalized requirements, serve as the basis for the scenario evolution process. As long as the adaptation space $\mathfrak{A}$ remains constant (and with it the system goals), system development should form an adaptation sequence. Consequently, scenario evolution should then form a scenario sequence for that adaptation sequence. This means (according to Definition 16), the scenario suite

Taken from original publication: Thomas Gabor, Marie Kiermeier, Andreas Sedlmeier, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, and Jan Wieghardt. Adapting quality assurance to adaptive systems: The scenario coevolution paradigm. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2018



is augmented with newly generated scenarios (for new system goals or just more specialized subgoals) or with scenarios with increased requirements on fitness.[5] Ideally, the scenario evolution process should lead the learning components on the right path towards the desired solution. The ability to re-assign fitness priorities allows for an arms race between adaptive system and scenario suite (see Concept 2).

Augmenting Requirements. Beyond requirements engineering, it is necessary to include knowledge that will be generated during training and learning by the adaptive components. Mainly, recognized scenarios that work well with early version of the adaptive system should be used as checks and tests when the system becomes more complex. This approach imitates the optimization technique of importance sampling on a systems engineering level. There are two central issues that need to be answered in this early phase of the development process:

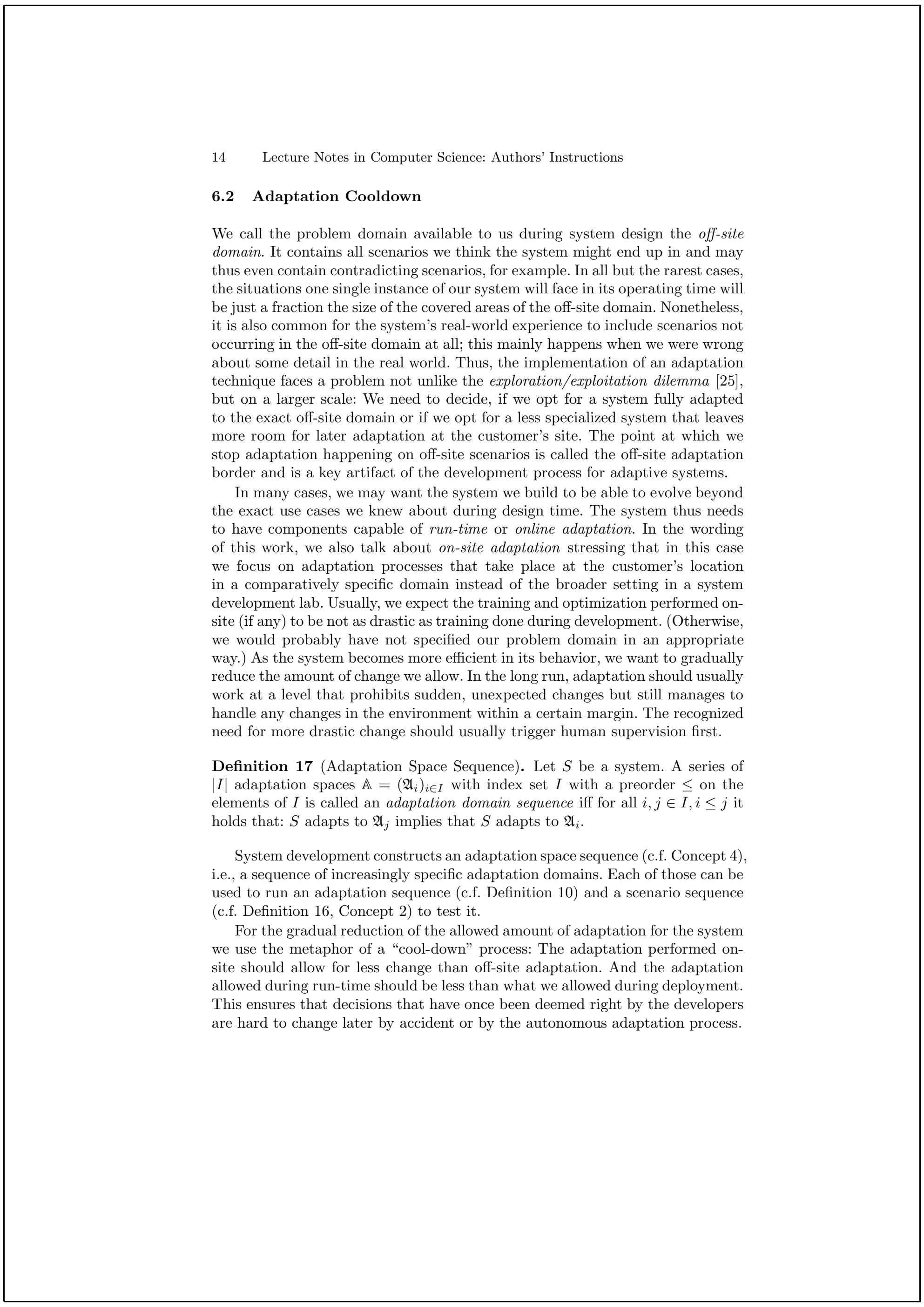
– Behavior Observation: How can system behavior be generated in a realistic manner? Are the formal specifications powerful enough? Can we employ human-labeled experience?
– Behavior Assessment: How can the quality of observed behavior be adequately assessed? Can we define a model for the users' intent? Can we employ human-labeled review?

Breaking Down Requirements. A central task of successful requirements engineering is to split up the use cases in atomic units that ideally describe singular features. In the dynamic world, we want to leave more room for adaptive system behavior. Thus, the requirements we formulate tend to be more general in notion. It is thus even more important to split them up in meaningful ways in order to derive new sets of scenarios. The following design axes (without any claim to completeness) may be found useful to break down requirements of adaptive systems:

– Scope and Locality: Can the goal be applied/checked locally or does it involve multiple components? Which components fall into the scope of the goal? Is emergent system behavior desirable or considered harmful?
– Decomposition and Smoothness: Can internal (possibly more specific) requirements be developed? Can the overall goal be composed from a clear set of subgoals? Can the goal function be smoothened, for example by providing intermediate goals? Can subgoal decomposition change dynamically via adaptation or is it structurally static?
– Uncertainty and Interaction: Are all goals given with full certainty? Is it possible to reason about the relative importance of goal fulfillment for specific goals a priori? Which dynamic goals have an interface with human users or other systems?

[5] Note that every change in $\mathfrak{A}$ starts new sequences.

Taken from original publication: Thomas Gabor, Marie Kiermeier, Andreas Sedlmeier, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, and Jan Wieghardt. Adapting quality assurance to adaptive systems: The scenario coevolution paradigm. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2018



6.2 Adaptation Cooldown

We call the problem domain available to us during system design the *off-site domain*. It contains all scenarios we think the system might end up in and may thus even contain contradicting scenarios, for example. In all but the rarest cases, the situations one single instance of our system will face in its operating time will be just a fraction the size of the covered areas of the off-site domain. Nonetheless, it is also common for the system's real-world experience to include scenarios not occurring in the off-site domain at all; this mainly happens when we were wrong about some detail in the real world. Thus, the implementation of an adaptation technique faces a problem not unlike the *exploration/exploitation dilemma* [25], but on a larger scale: We need to decide, if we opt for a system fully adapted to the exact off-site domain or if we opt for a less specialized system that leaves more room for later adaptation at the customer's site. The point at which we stop adaptation happening on off-site scenarios is called the off-site adaptation border and is a key artifact of the development process for adaptive systems.

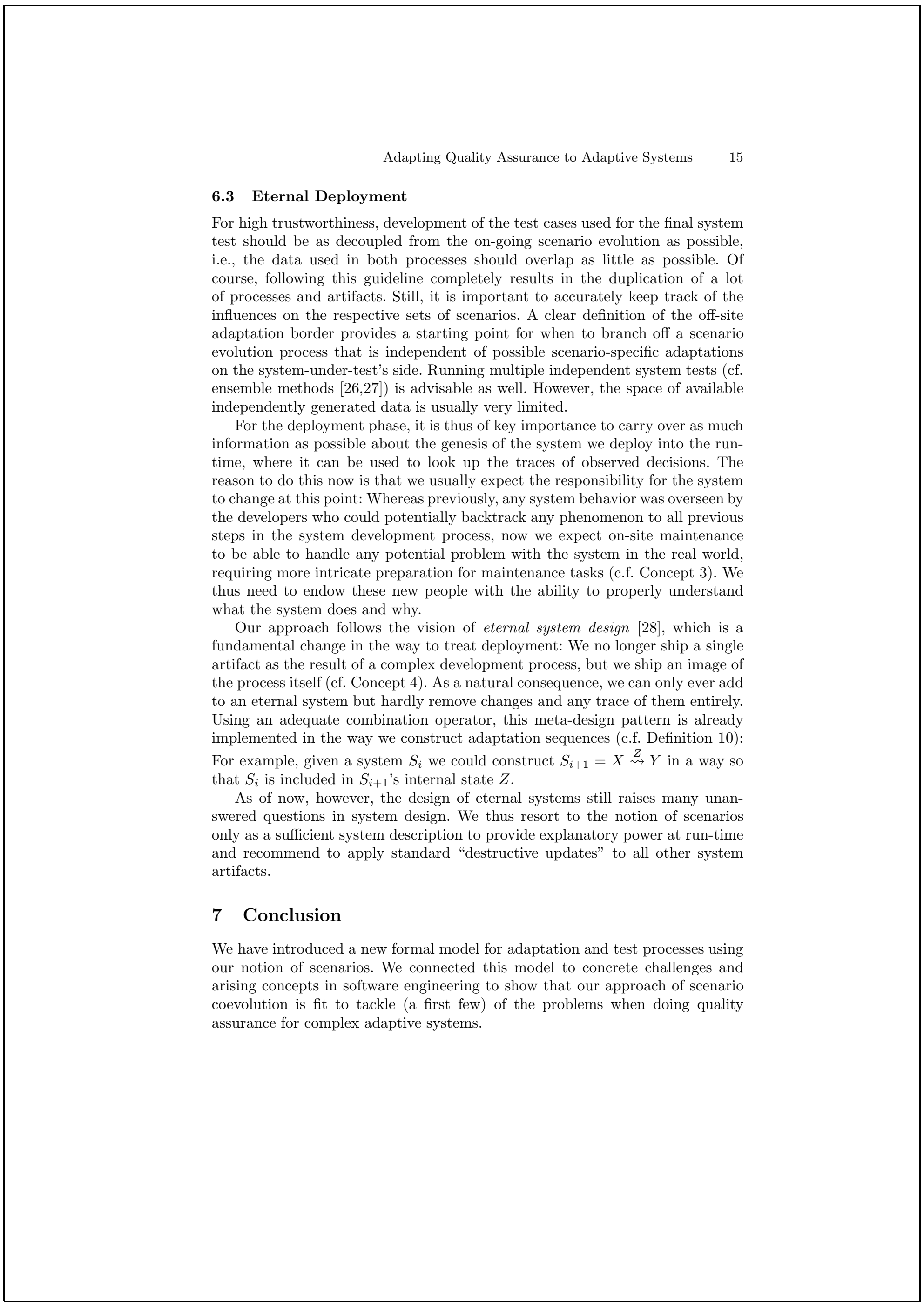
In many cases, we may want the system we build to be able to evolve beyond the exact use cases we knew about during design time. The system thus needs to have components capable of *run-time* or *online adaptation*. In the wording of this work, we also talk about *on-site adaptation* stressing that in this case we focus on adaptation processes that take place at the customer's location in a comparatively specific domain instead of the broader setting in a system development lab. Usually, we expect the training and optimization performed on-site (if any) to be not as drastic as training done during development. (Otherwise, we would probably have not specified our problem domain in an appropriate way.) As the system becomes more efficient in its behavior, we want to gradually reduce the amount of change we allow. In the long run, adaptation should usually work at a level that prohibits sudden, unexpected changes but still manages to handle any changes in the environment within a certain margin. The recognized need for more drastic change should usually trigger human supervision first.

Definition 17 (Adaptation Space Sequence)**.** Let S be a system. A series of $|I|$ adaptation spaces $\mathbb{A} = (\mathfrak{A}_i)_{i \in I}$ with index set I with a preorder $\leq$ on the elements of I is called an *adaptation domain sequence* iff for all $i, j \in I, i \leq j$ it holds that: S adapts to $\mathfrak{A}_j$ implies that S adapts to $\mathfrak{A}_i$.

System development constructs an adaptation space sequence (c.f. Concept 4), i.e., a sequence of increasingly specific adaptation domains. Each of those can be used to run an adaptation sequence (c.f. Definition 10) and a scenario sequence (c.f. Definition 16, Concept 2) to test it.

For the gradual reduction of the allowed amount of adaptation for the system we use the metaphor of a "cool-down" process: The adaptation performed on-site should allow for less change than off-site adaptation. And the adaptation allowed during run-time should be less than what we allowed during deployment. This ensures that decisions that have once been deemed right by the developers are hard to change later by accident or by the autonomous adaptation process.

Taken from original publication: Thomas Gabor, Marie Kiermeier, Andreas Sedlmeier, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, and Jan Wieghardt. Adapting quality assurance to adaptive systems: The scenario coevolution paradigm. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2018



6.3 Eternal Deployment

For high trustworthiness, development of the test cases used for the final system test should be as decoupled from the on-going scenario evolution as possible, i.e., the data used in both processes should overlap as little as possible. Of course, following this guideline completely results in the duplication of a lot of processes and artifacts. Still, it is important to accurately keep track of the influences on the respective sets of scenarios. A clear definition of the off-site adaptation border provides a starting point for when to branch off a scenario evolution process that is independent of possible scenario-specific adaptations on the system-under-test's side. Running multiple independent system tests (cf. ensemble methods [26,27]) is advisable as well. However, the space of available independently generated data is usually very limited.

For the deployment phase, it is thus of key importance to carry over as much information as possible about the genesis of the system we deploy into the run-time, where it can be used to look up the traces of observed decisions. The reason to do this now is that we usually expect the responsibility for the system to change at this point: Whereas previously, any system behavior was overseen by the developers who could potentially backtrack any phenomenon to all previous steps in the system development process, now we expect on-site maintenance to be able to handle any potential problem with the system in the real world, requiring more intricate preparation for maintenance tasks (c.f. Concept 3). We thus need to endow these new people with the ability to properly understand what the system does and why.

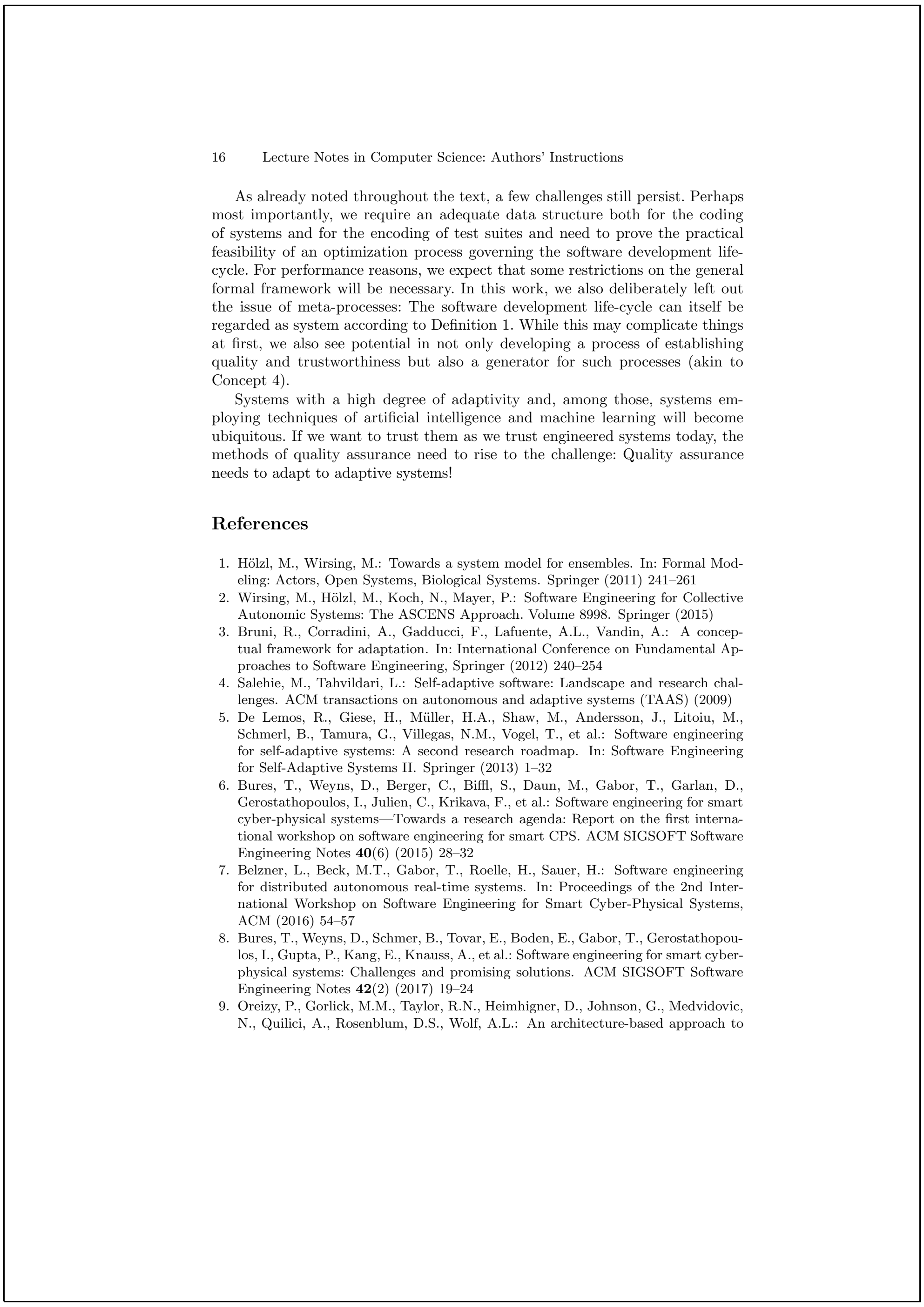
Our approach follows the vision of *eternal system design* [28], which is a fundamental change in the way to treat deployment: We no longer ship a single artifact as the result of a complex development process, but we ship an image of the process itself (cf. Concept 4). As a natural consequence, we can only ever add to an eternal system but hardly remove changes and any trace of them entirely. Using an adequate combination operator, this meta-design pattern is already implemented in the way we construct adaptation sequences (c.f. Definition 10): For example, given a system S_i we could construct $S_{i+1} = X \overset{Z}{\rightsquigarrow} Y$ in a way so that S_i is included in S_{i+1}'s internal state Z.

As of now, however, the design of eternal systems still raises many unanswered questions in system design. We thus resort to the notion of scenarios only as a sufficient system description to provide explanatory power at run-time and recommend to apply standard "destructive updates" to all other system artifacts.

7 Conclusion

We have introduced a new formal model for adaptation and test processes using our notion of scenarios. We connected this model to concrete challenges and arising concepts in software engineering to show that our approach of scenario coevolution is fit to tackle (a first few) of the problems when doing quality assurance for complex adaptive systems.

Taken from original publication: Thomas Gabor, Marie Kiermeier, Andreas Sedlmeier, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, and Jan Wieghardt. Adapting quality assurance to adaptive systems: The scenario coevolution paradigm. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2018



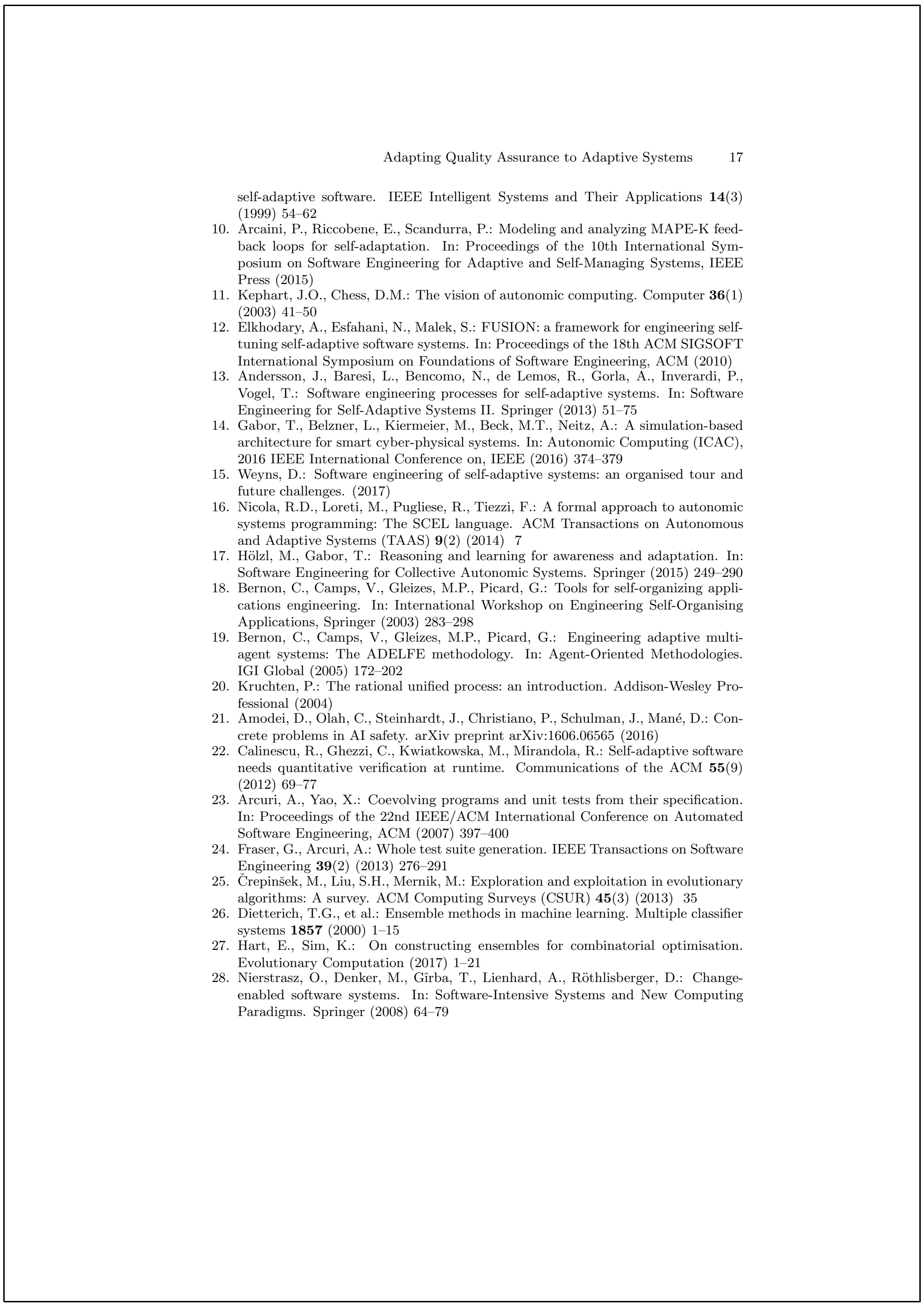
As already noted throughout the text, a few challenges still persist. Perhaps most importantly, we require an adequate data structure both for the coding of systems and for the encoding of test suites and need to prove the practical feasibility of an optimization process governing the software development life-cycle. For performance reasons, we expect that some restrictions on the general formal framework will be necessary. In this work, we also deliberately left out the issue of meta-processes: The software development life-cycle can itself be regarded as system according to Definition 1. While this may complicate things at first, we also see potential in not only developing a process of establishing quality and trustworthiness but also a generator for such processes (akin to Concept 4).

Systems with a high degree of adaptivity and, among those, systems employing techniques of artificial intelligence and machine learning will become ubiquitous. If we want to trust them as we trust engineered systems today, the methods of quality assurance need to rise to the challenge: Quality assurance needs to adapt to adaptive systems!

References

1. Hölzl, M., Wirsing, M.: Towards a system model for ensembles. In: Formal Modeling: Actors, Open Systems, Biological Systems. Springer (2011) 241–261
2. Wirsing, M., Hölzl, M., Koch, N., Mayer, P.: Software Engineering for Collective Autonomic Systems: The ASCENS Approach. Volume 8998. Springer (2015)
3. Bruni, R., Corradini, A., Gadducci, F., Lafuente, A.L., Vandin, A.: A conceptual framework for adaptation. In: International Conference on Fundamental Approaches to Software Engineering, Springer (2012) 240–254
4. Salehie, M., Tahvildari, L.: Self-adaptive software: Landscape and research challenges. ACM transactions on autonomous and adaptive systems (TAAS) (2009)
5. De Lemos, R., Giese, H., Müller, H.A., Shaw, M., Andersson, J., Litoiu, M., Schmerl, B., Tamura, G., Villegas, N.M., Vogel, T., et al.: Software engineering for self-adaptive systems: A second research roadmap. In: Software Engineering for Self-Adaptive Systems II. Springer (2013) 1–32
6. Bures, T., Weyns, D., Berger, C., Biffl, S., Daun, M., Gabor, T., Garlan, D., Gerostathopoulos, I., Julien, C., Krikava, F., et al.: Software engineering for smart cyber-physical systems—Towards a research agenda: Report on the first international workshop on software engineering for smart CPS. ACM SIGSOFT Software Engineering Notes **40**(6) (2015) 28–32
7. Belzner, L., Beck, M.T., Gabor, T., Roelle, H., Sauer, H.: Software engineering for distributed autonomous real-time systems. In: Proceedings of the 2nd International Workshop on Software Engineering for Smart Cyber-Physical Systems, ACM (2016) 54–57
8. Bures, T., Weyns, D., Schmer, B., Tovar, E., Boden, E., Gabor, T., Gerostathopoulos, I., Gupta, P., Kang, E., Knauss, A., et al.: Software engineering for smart cyber-physical systems: Challenges and promising solutions. ACM SIGSOFT Software Engineering Notes **42**(2) (2017) 19–24
9. Oreizy, P., Gorlick, M.M., Taylor, R.N., Heimhigner, D., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D.S., Wolf, A.L.: An architecture-based approach to

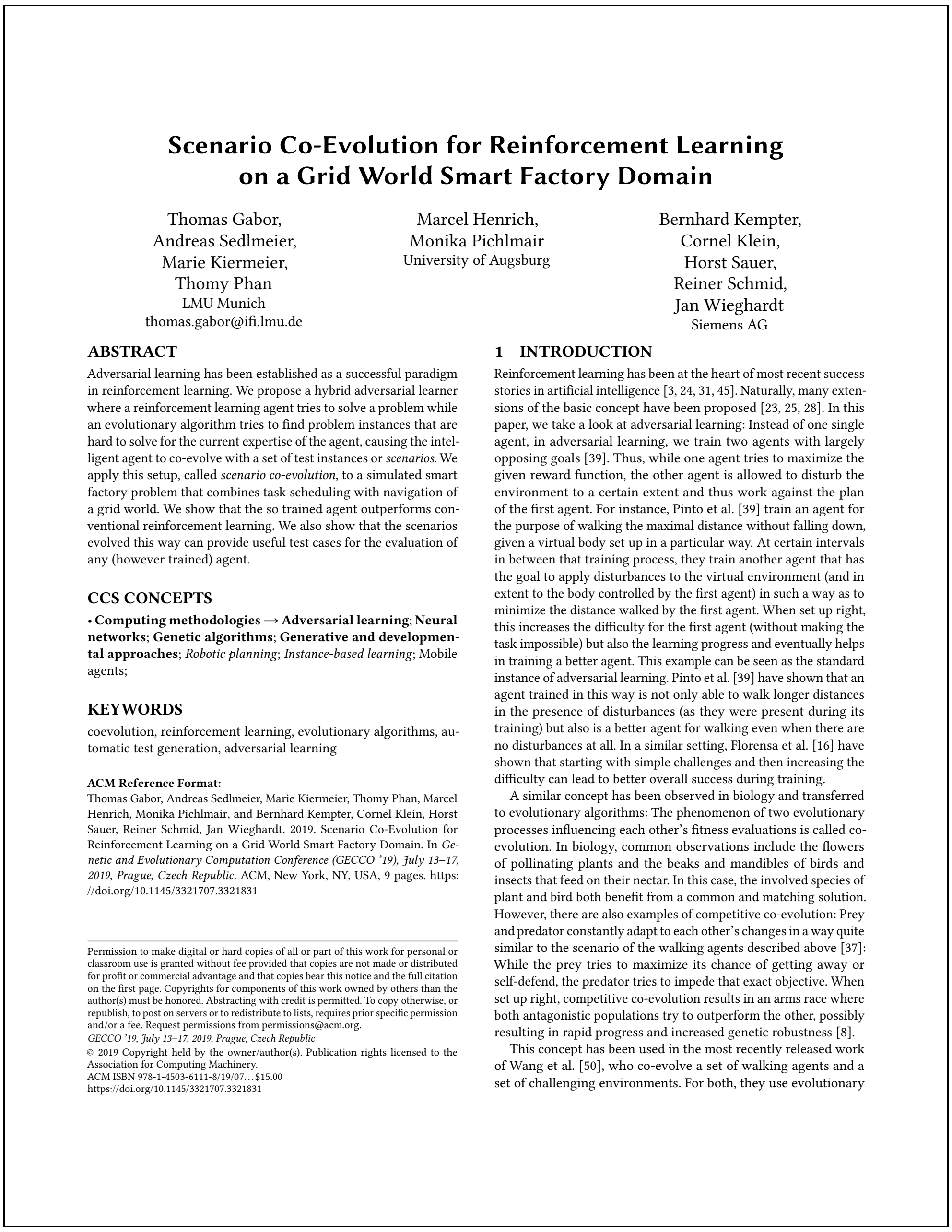
Taken from original publication: Thomas Gabor, Marie Kiermeier, Andreas Sedlmeier, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, and Jan Wieghardt. Adapting quality assurance to adaptive systems: The scenario coevolution paradigm. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2018



self-adaptive software. IEEE Intelligent Systems and Their Applications **14**(3) (1999) 54–62

10. Arcaini, P., Riccobene, E., Scandurra, P.: Modeling and analyzing MAPE-K feedback loops for self-adaptation. In: Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, IEEE Press (2015)
11. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer **36**(1) (2003) 41–50
12. Elkhodary, A., Esfahani, N., Malek, S.: FUSION: a framework for engineering self-tuning self-adaptive software systems. In: Proceedings of the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering, ACM (2010)
13. Andersson, J., Baresi, L., Bencomo, N., de Lemos, R., Gorla, A., Inverardi, P., Vogel, T.: Software engineering processes for self-adaptive systems. In: Software Engineering for Self-Adaptive Systems II. Springer (2013) 51–75
14. Gabor, T., Belzner, L., Kiermeier, M., Beck, M.T., Neitz, A.: A simulation-based architecture for smart cyber-physical systems. In: Autonomic Computing (ICAC), 2016 IEEE International Conference on, IEEE (2016) 374–379
15. Weyns, D.: Software engineering of self-adaptive systems: an organised tour and future challenges. (2017)
16. Nicola, R.D., Loreti, M., Pugliese, R., Tiezzi, F.: A formal approach to autonomic systems programming: The SCEL language. ACM Transactions on Autonomous and Adaptive Systems (TAAS) **9**(2) (2014) 7
17. Hölzl, M., Gabor, T.: Reasoning and learning for awareness and adaptation. In: Software Engineering for Collective Autonomic Systems. Springer (2015) 249–290
18. Bernon, C., Camps, V., Gleizes, M.P., Picard, G.: Tools for self-organizing applications engineering. In: International Workshop on Engineering Self-Organising Applications, Springer (2003) 283–298
19. Bernon, C., Camps, V., Gleizes, M.P., Picard, G.: Engineering adaptive multi-agent systems: The ADELFE methodology. In: Agent-Oriented Methodologies. IGI Global (2005) 172–202
20. Kruchten, P.: The rational unified process: an introduction. Addison-Wesley Professional (2004)
21. Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., Mané, D.: Concrete problems in AI safety. arXiv preprint arXiv:1606.06565 (2016)
22. Calinescu, R., Ghezzi, C., Kwiatkowska, M., Mirandola, R.: Self-adaptive software needs quantitative verification at runtime. Communications of the ACM **55**(9) (2012) 69–77
23. Arcuri, A., Yao, X.: Coevolving programs and unit tests from their specification. In: Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering, ACM (2007) 397–400
24. Fraser, G., Arcuri, A.: Whole test suite generation. IEEE Transactions on Software Engineering **39**(2) (2013) 276–291
25. Črepinšek, M., Liu, S.H., Mernik, M.: Exploration and exploitation in evolutionary algorithms: A survey. ACM Computing Surveys (CSUR) **45**(3) (2013) 35
26. Dietterich, T.G., et al.: Ensemble methods in machine learning. Multiple classifier systems **1857** (2000) 1–15
27. Hart, E., Sim, K.: On constructing ensembles for combinatorial optimisation. Evolutionary Computation (2017) 1–21
28. Nierstrasz, O., Denker, M., Gîrba, T., Lienhard, A., Röthlisberger, D.: Change-enabled software systems. In: Software-Intensive Systems and New Computing Paradigms. Springer (2008) 64–79

Taken from original publication: Thomas Gabor, Andreas Sedlmeier, Marie Kiermeier, Thomy Phan, Marcel Henrich, Monika Pichlmair, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, and Jan Wieghardt. Scenario co-evolution for reinforcement learning on a grid world smart factory domain. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 898–906, 2019



Scenario Co-Evolution for Reinforcement Learning on a Grid World Smart Factory Domain

Thomas Gabor,
Andreas Sedlmeier,
Marie Kiermeier,
Thomy Phan
LMU Munich
thomas.gabor@ifi.lmu.de

Marcel Henrich,
Monika Pichlmair
University of Augsburg

Bernhard Kempter,
Cornel Klein,
Horst Sauer,
Reiner Schmid,
Jan Wieghardt
Siemens AG

ABSTRACT

Adversarial learning has been established as a successful paradigm in reinforcement learning. We propose a hybrid adversarial learner where a reinforcement learning agent tries to solve a problem while an evolutionary algorithm tries to find problem instances that are hard to solve for the current expertise of the agent, causing the intelligent agent to co-evolve with a set of test instances or *scenarios*. We apply this setup, called *scenario co-evolution*, to a simulated smart factory problem that combines task scheduling with navigation of a grid world. We show that the so trained agent outperforms conventional reinforcement learning. We also show that the scenarios evolved this way can provide useful test cases for the evaluation of any (however trained) agent.

CCS CONCEPTS

• **Computing methodologies → Adversarial learning**; **Neural networks**; **Genetic algorithms**; **Generative and developmental approaches**; *Robotic planning*; *Instance-based learning*; Mobile agents;

KEYWORDS

coevolution, reinforcement learning, evolutionary algorithms, automatic test generation, adversarial learning

ACM Reference Format:
Thomas Gabor, Andreas Sedlmeier, Marie Kiermeier, Thomy Phan, Marcel Henrich, Monika Pichlmair, and Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, Jan Wieghardt. 2019. Scenario Co-Evolution for Reinforcement Learning on a Grid World Smart Factory Domain. In *Genetic and Evolutionary Computation Conference (GECCO '19), July 13–17, 2019, Prague, Czech Republic.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3321707.3321831

1 INTRODUCTION

Reinforcement learning has been at the heart of most recent success stories in artificial intelligence [3, 24, 31, 45]. Naturally, many extensions of the basic concept have been proposed [23, 25, 28]. In this paper, we take a look at adversarial learning: Instead of one single agent, in adversarial learning, we train two agents with largely opposing goals [39]. Thus, while one agent tries to maximize the given reward function, the other agent is allowed to disturb the environment to a certain extent and thus work against the plan of the first agent. For instance, Pinto et al. [39] train an agent for the purpose of walking the maximal distance without falling down, given a virtual body set up in a particular way. At certain intervals in between that training process, they train another agent that has the goal to apply disturbances to the virtual environment (and in extent to the body controlled by the first agent) in such a way as to minimize the distance walked by the first agent. When set up right, this increases the difficulty for the first agent (without making the task impossible) but also the learning progress and eventually helps in training a better agent. This example can be seen as the standard instance of adversarial learning. Pinto et al. [39] have shown that an agent trained in this way is not only able to walk longer distances in the presence of disturbances (as they were present during its training) but also is a better agent for walking even when there are no disturbances at all. In a similar setting, Florensa et al. [16] have shown that starting with simple challenges and then increasing the difficulty can lead to better overall success during training.

A similar concept has been observed in biology and transferred to evolutionary algorithms: The phenomenon of two evolutionary processes influencing each other's fitness evaluations is called co-evolution. In biology, common observations include the flowers of pollinating plants and the beaks and mandibles of birds and insects that feed on their nectar. In this case, the involved species of plant and bird both benefit from a common and matching solution. However, there are also examples of competitive co-evolution: Prey and predator constantly adapt to each other's changes in a way quite similar to the scenario of the walking agents described above [37]: While the prey tries to maximize its chance of getting away or self-defend, the predator tries to impede that exact objective. When set up right, competitive co-evolution results in an arms race where both antagonistic populations try to outperform the other, possibly resulting in rapid progress and increased genetic robustness [8].

This concept has been used in the most recently released work of Wang et al. [50], who co-evolve a set of walking agents and a set of challenging environments. For both, they use evolutionary

Taken from original publication: Thomas Gabor, Andreas Sedlmeier, Marie Kiermeier, Thomy Phan, Marcel Henrich, Monika Pichlmair, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, and Jan Wieghardt. Scenario co-evolution for reinforcement learning on a grid world smart factory domain. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 898–906, 2019

strategies, which are related to some techniques of reinforcement learning [54] but are now rather seen as a method of gradient-based black-box optimization in contrast to classical reinforcement learning [43, 53]. In their approach, called POET (which stands for Paired Open-Ended Trailblazer), they pair up instances of agents and environments to form single individuals in an overarching evolutionary process. By contrast, we suggest (among other differences, cf. Section 3) a simpler interaction model for agents and environments and train a reinforcement learning agent using standard back-propagation. Still, we are able to produce similar results for the discrete smart factory domain we introduce in this paper.

In all three examples introduced so far, the antagonistic agents or species only interact via the mutually shared results of fitness evaluations: A good result for one side is bad for the the other. (We will formalize this setup in Section 4.) This allows us to combine various techniques here, i.e., we propose to use a reinforcement learning agent to train for certain behavior (that is well encoded using neural networks) and an evolutionary algorithm to evolve challenging environments to operate in (that are easily encoded using discrete vectors without an obvious gradient to them). Thus, we can use adequate data structures on both sides of the co-evolution. Motivated by the industry origin of the domain, we evaluate our results not only against the fitness score, but also against a criterium of solved/failed instances, showing greater robustness for the co-evolutionary approach.

We consider this approach an instance of the general architectural model we described in [22], hence we also call it *scenario co-evolution*. There, the co-evolutionary principle is applied to software engineering, where productive code and software tests co-evolve: A test is good when it finds bad behavior in the productive code, while productive code is good when no test finds bad behavior. As suggested in both [50] and [22], we show in this paper that the co-evolved environments (also called *scenarios*) can be used efficiently for classical software testing, outperforming random testing in their prowess to challenge an agent.

We now continue to formally introduce the basic concepts mentioned so far (cf. Section 2) and then give an overview of related work (cf. Section 3). We formally describe our approach at scenario co-evolution in Section 4 and provide an empirical evaluation in Section 5. We conclude with Section 6.

2 BASICS

2.1 Markov Decision Processes

We base our problem formulation on the notion of a Markov decision process (MDP) [41], which is given via the tuple: $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$. $\mathcal{S}$ is a (finite) set of states; $s_t \in \mathcal{S}$ is the state of the MDP at time step t. $\mathcal{A}$ is the (finite) set of actions; $a_t \in \mathcal{A}$ is the action the MDP takes at time step t. $\mathcal{P}(s_{t+1}|s_t, a_t)$ is the transition probability function; a state transition occurs by executing an action a_t in a state s_t. The resulting next state s_{t+1} is then determined according to $\mathcal{P}$. Note that in this paper we focus on a deterministic domain represented by a deterministic MDP, so $\mathcal{P}(s_{t+1}|s_t, a_t) \in \{0, 1\}$. Finally, $\mathcal{R}(s_t, a_t)$ is the reward awarded when the MDP takes action a_t when in state s_t; for this paper we assume that $\mathcal{R}(s_t, a_t) \in \mathbb{R}$.

The goal is to find a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ in the space of all possible policies Π, which maximizes the (discounted) return G_t at state s_t over a potentially infinite horizon, given via

$$G_t = \sum_{k=0}^{\infty} \gamma^k \cdot \mathcal{R}(s_{t+k}, a_{t+k}) \tag{1}$$

where $\gamma \in [0, 1]$ is the discount factor.

2.2 Reinforcement Learning

In order to search the policy space Π, we consider model-free reinforcement learning (RL), in which an agent interacts with an environment given as an MDP $\mathcal{M}$ by executing a sequence of actions $a_t \in \mathcal{A}, t = 0, 1, \ldots$ [48]. In the fully observable case of reinforcement learning, the agent knows its current state s_t and the action space $\mathcal{A}$, but not the effect of executing a_t in s_t, i.e., $\mathcal{P}(s_{t+1}|s_t, a_t)$ and $\mathcal{R}(s_t, a_t)$. In order to find the optimal policy π^* a commonly used value-based approach is Q-Learning [51], named for the action-value function $Q^\pi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}, \pi \in \Pi$, which describes the expected accumulated reward $Q^\pi(s_t, a_t)$ when taking action a_t when in state s_t and then following the policy π for all states $s_{t+1}, s_{t+2}, \ldots$ afterwards.

The optimal action-value function Q^* is any action-value function that yields higher accumulated rewards than all other action-value functions, i.e., $Q^*(s_t, a_t) \geq Q^\pi(s_t, a_t)\ \forall \pi \in \Pi$. Q-Learning aims to approximate Q^* by starting from an initial guess for Q, which is then updated via

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{2}$$

by making use of experience samples $e_t = (s_t, a_t, s_{t+1}, r_t)$, where r_t is the reward earned at time step t, i.e., by executing action a_t when in state s_t. The learning rate α is a usually setup-specific parameter.

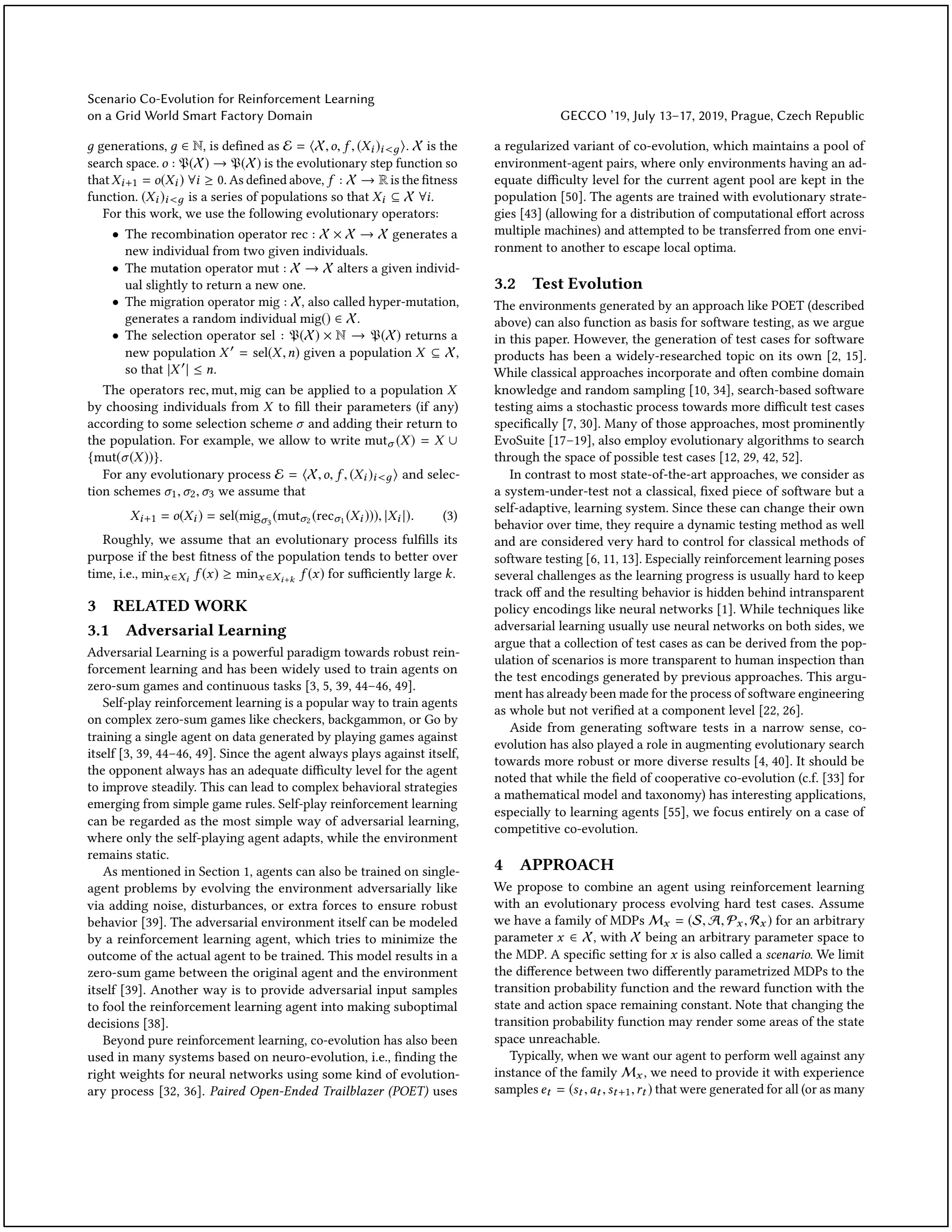
The learned action-value function Q converges to the optimal action-value function Q^*, which then implies an optimal policy $\pi^*(s_t) = \arg\max_a Q(s_t, a)$.

It is common to use a parameterized function approximator (like a neural network), to approximate the action-value function: $Q(s_t, a_t; \theta) \approx Q^*(s_t, a_t)$ with θ specifying the weights of the neural network. When a deep neural network is used as the function approximator, this approach is called deep reinforcement learning. Mnih et al. [31] showed that combining this approach with deep convolutional networks allows for successful learning from high-dimensional input features like raw image data.

2.3 Evolutionary Algorithms

For this paper, we assume an evolutionary process (EP) to be defined as follows: Given a fitness function $f : \mathcal{X} \rightarrow \mathbb{R}$ for an arbitrary set $\mathcal{X}$ called the search space, we want to find an individual $x \in \mathcal{X}$ with the best fitness, i.e., $f(x) \leq f(x')\ \forall x' \in \mathcal{X}$. Note that for consistency with the later application, we assume that the best fitness has the lowest values, i.e., that we try to minimize the fitness values. Usually, the search space $\mathcal{X}$ is too large or too complicated to guarantee that we can find the exact best individual(s) using standard computing models (and physically realistic time). Thus, we take discrete subsets of the search space $\mathcal{X}$ via sampling and iteratively improve their fitness. An evolutionary process $\mathcal{E}$ over

Taken from original publication: Thomas Gabor, Andreas Sedlmeier, Marie Kiermeier, Thomy Phan, Marcel Henrich, Monika Pichlmair, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, and Jan Wieghardt. Scenario co-evolution for reinforcement learning on a grid world smart factory domain. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 898–906, 2019



g generations, $g \in \mathbb{N}$, is defined as $\mathcal{E} = \langle \mathcal{X}, o, f, (X_i)_{i<g} \rangle$. $\mathcal{X}$ is the search space. $o : \mathfrak{P}(\mathcal{X}) \to \mathfrak{P}(\mathcal{X})$ is the evolutionary step function so that $X_{i+1} = o(X_i)\ \forall i \geq 0$. As defined above, $f : \mathcal{X} \to \mathbb{R}$ is the fitness function. $(X_i)_{i<g}$ is a series of populations so that $X_i \subseteq \mathcal{X}\ \forall i$.

For this work, we use the following evolutionary operators:

- The recombination operator $\mathrm{rec} : \mathcal{X} \times \mathcal{X} \to \mathcal{X}$ generates a new individual from two given individuals.
- The mutation operator $\mathrm{mut} : \mathcal{X} \to \mathcal{X}$ alters a given individual slightly to return a new one.
- The migration operator $\mathrm{mig} : \mathcal{X}$, also called hyper-mutation, generates a random individual $\mathrm{mig}() \in \mathcal{X}$.
- The selection operator $\mathrm{sel} : \mathfrak{P}(\mathcal{X}) \times \mathbb{N} \to \mathfrak{P}(\mathcal{X})$ returns a new population $X' = \mathrm{sel}(X, n)$ given a population $X \subseteq \mathcal{X}$, so that $|X'| \leq n$.

The operators $\mathrm{rec}, \mathrm{mut}, \mathrm{mig}$ can be applied to a population X by choosing individuals from X to fill their parameters (if any) according to some selection scheme σ and adding their return to the population. For example, we allow to write $\mathrm{mut}_\sigma(X) = X \cup \{\mathrm{mut}(\sigma(X))\}$.

For any evolutionary process $\mathcal{E} = \langle \mathcal{X}, o, f, (X_i)_{i<g} \rangle$ and selection schemes $\sigma_1, \sigma_2, \sigma_3$ we assume that

$$X_{i+1} = o(X_i) = \mathrm{sel}(\mathrm{mig}_{\sigma_3}(\mathrm{mut}_{\sigma_2}(\mathrm{rec}_{\sigma_1}(X_i))), |X_i|). \qquad (3)$$

Roughly, we assume that an evolutionary process fulfills its purpose if the best fitness of the population tends to better over time, i.e., $\min_{x \in X_i} f(x) \geq \min_{x \in X_{i+k}} f(x)$ for sufficiently large k.

3 RELATED WORK

3.1 Adversarial Learning

Adversarial Learning is a powerful paradigm towards robust reinforcement learning and has been widely used to train agents on zero-sum games and continuous tasks [3, 5, 39, 44–46, 49].

Self-play reinforcement learning is a popular way to train agents on complex zero-sum games like checkers, backgammon, or Go by training a single agent on data generated by playing games against itself [3, 39, 44–46, 49]. Since the agent always plays against itself, the opponent always has an adequate difficulty level for the agent to improve steadily. This can lead to complex behavioral strategies emerging from simple game rules. Self-play reinforcement learning can be regarded as the most simple way of adversarial learning, where only the self-playing agent adapts, while the environment remains static.

As mentioned in Section 1, agents can also be trained on single-agent problems by evolving the environment adversarially like via adding noise, disturbances, or extra forces to ensure robust behavior [39]. The adversarial environment itself can be modeled by a reinforcement learning agent, which tries to minimize the outcome of the actual agent to be trained. This model results in a zero-sum game between the original agent and the environment itself [39]. Another way is to provide adversarial input samples to fool the reinforcement learning agent into making suboptimal decisions [38].

Beyond pure reinforcement learning, co-evolution has also been used in many systems based on neuro-evolution, i.e., finding the right weights for neural networks using some kind of evolutionary process [32, 36]. *Paired Open-Ended Trailblazer (POET)* uses a regularized variant of co-evolution, which maintains a pool of environment-agent pairs, where only environments having an adequate difficulty level for the current agent pool are kept in the population [50]. The agents are trained with evolutionary strategies [43] (allowing for a distribution of computational effort across multiple machines) and attempted to be transferred from one environment to another to escape local optima.

3.2 Test Evolution

The environments generated by an approach like POET (described above) can also function as basis for software testing, as we argue in this paper. However, the generation of test cases for software products has been a widely-researched topic on its own [2, 15]. While classical approaches incorporate and often combine domain knowledge and random sampling [10, 34], search-based software testing aims a stochastic process towards more difficult test cases specifically [7, 30]. Many of those approaches, most prominently EvoSuite [17–19], also employ evolutionary algorithms to search through the space of possible test cases [12, 29, 42, 52].

In contrast to most state-of-the-art approaches, we consider as a system-under-test not a classical, fixed piece of software but a self-adaptive, learning system. Since these can change their own behavior over time, they require a dynamic testing method as well and are considered very hard to control for classical methods of software testing [6, 11, 13]. Especially reinforcement learning poses several challenges as the learning progress is usually hard to keep track off and the resulting behavior is hidden behind intransparent policy encodings like neural networks [1]. While techniques like adversarial learning usually use neural networks on both sides, we argue that a collection of test cases as can be derived from the population of scenarios is more transparent to human inspection than the test encodings generated by previous approaches. This argument has already been made for the process of software engineering as whole but not verified at a component level [22, 26].

Aside from generating software tests in a narrow sense, co-evolution has also played a role in augmenting evolutionary search towards more robust or more diverse results [4, 40]. It should be noted that while the field of cooperative co-evolution (c.f. [33] for a mathematical model and taxonomy) has interesting applications, especially to learning agents [55], we focus entirely on a case of competitive co-evolution.

4 APPROACH

We propose to combine an agent using reinforcement learning with an evolutionary process evolving hard test cases. Assume we have a family of MDPs $\mathcal{M}_x = (\mathcal{S}, \mathcal{A}, \mathcal{P}_x, \mathcal{R}_x)$ for an arbitrary parameter $x \in \mathcal{X}$, with $\mathcal{X}$ being an arbitrary parameter space to the MDP. A specific setting for x is also called a *scenario*. We limit the difference between two differently parametrized MDPs to the transition probability function and the reward function with the state and action space remaining constant. Note that changing the transition probability function may render some areas of the state space unreachable.

Typically, when we want our agent to perform well against any instance of the family $\mathcal{M}_x$, we need to provide it with experience samples $e_t = (s_t, a_t, s_{t+1}, r_t)$ that were generated for all (or as many

Taken from original publication: Thomas Gabor, Andreas Sedlmeier, Marie Kiermeier, Thomy Phan, Marcel Henrich, Monika Pichlmair, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, and Jan Wieghardt. Scenario co-evolution for reinforcement learning on a grid world smart factory domain. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 898–906, 2019

as possible) different scenarios $x \in \mathcal{X}$. Note that the setting of x directly affects the values of s_{t+1} via $\mathcal{P}_x$ and r_t via $\mathcal{R}_x$. Usually, some effect of a specific setting of x may also be visible in s_t and thus exposed to the agent. This is the case for the smart factory domain we introduce later.

The acknowledgement of certain non-user-controllable parameters within the environment is crucial to realistic applications. In most cases we have but a rough model of how the environment may behave but no way to pinpoint the specifics unless we try out all possibilities. So the agent needs to be trained against all of them, ideally. Of course, for sufficiently large or complex $\mathcal{X}$ this becomes infeasible. A standard approach is to take random samples from $\mathcal{X}$ instead. This causes the agent to specialize on the average scenario $x \in \mathcal{X}$ after a while of training, which may be a good choice per se. However, in most real-world scenarios, good average case performance is largely outweighed by bad worst case performance, i.e., a navigation software that (even rarely) provokes incidents is bad for the job, even if on average it finds the way quicker than its competition.

So instead of taking random samples from the scenario space $\mathcal{X}$, we may want to focus on the hard settings for x, i.e., those values x for which the agent's performance deteriorates. In order to do so, we have to find the respective values for x first, though. We propose to do so using an evolutionary algorithm (as described in Section 2.3) that optimizes for hard settings for x. This evolutionary algorithm constructs an evolutionary process with search space $\mathcal{X}$ (rendering our variable naming scheme consistent). The resulting population after a few generations of optimizing for hard x is then used to generate experience samples for the reinforcement learning agent. The best reinforcement learning agent so far is in turn used to evaluate the hardness of the settings for x for the next few generations of evolution.

Figure 1 shows a schematic representation of the combined process called *scenario co-evolution (SCoE)*: The interaction points between the evolutionary process and the reinforcement learning agent are:

- The experience samples necessary to train the agent are drawn using settings for $x \in \mathcal{X}$ that are included in the current population X of the evolutionary process. When all $x \in X$ have been used, the evolutionary process evolves further for a few generations.
- The fitness $f(x)$ assigned to each $x \in X$ is computed using the accumulated reward of running the current agent policy π on the MDP $\mathcal{M}_x$, i.e.,
$$f(x) = \sum_{t=0}^{h} \mathcal{R}_x(s_t, \pi(s_t)) \tag{4}$$
where h is the end of the current episode, i.e., $\mathcal{P}_x(s|s_h, a) = 0 \; \forall s \in \mathcal{S}, a \in \mathcal{A}$. Note that we defined the reinforcement learning agent to maximize its reward while the evolutionary process tries to minimize the fitness.

These interactions suffice to give rise to competitive co-evolution between a supposedly robust agent and a set of hard scenarios. However, our evaluation shows that the so-trained reinforcement learning agent not only performs better in the hard scenarios it was trained for, but also in randomly selected average scenarios. We call this the “exam effect”: When we confront the agent with hard scenarios (and it can solve those), we can also assume it can solve easy scenarios. Thus, there is no additional use to confront it with easy scenarios during training. Effectively, this is why we can talk about “easy” and “hard” scenarios in the first place: The agent does not simply specialize on a specific subset of scenarios and gets worse on other scenarios in return, but it gets better in all scenarios by training on some scenarios we thus call “hard”. This implies a hierarchy or order among scenarios. The scenarios that can be learned alongside training on hard scenarios can then be called “easy”.

5 EVALUATION

5.1 Smart Factory Domain

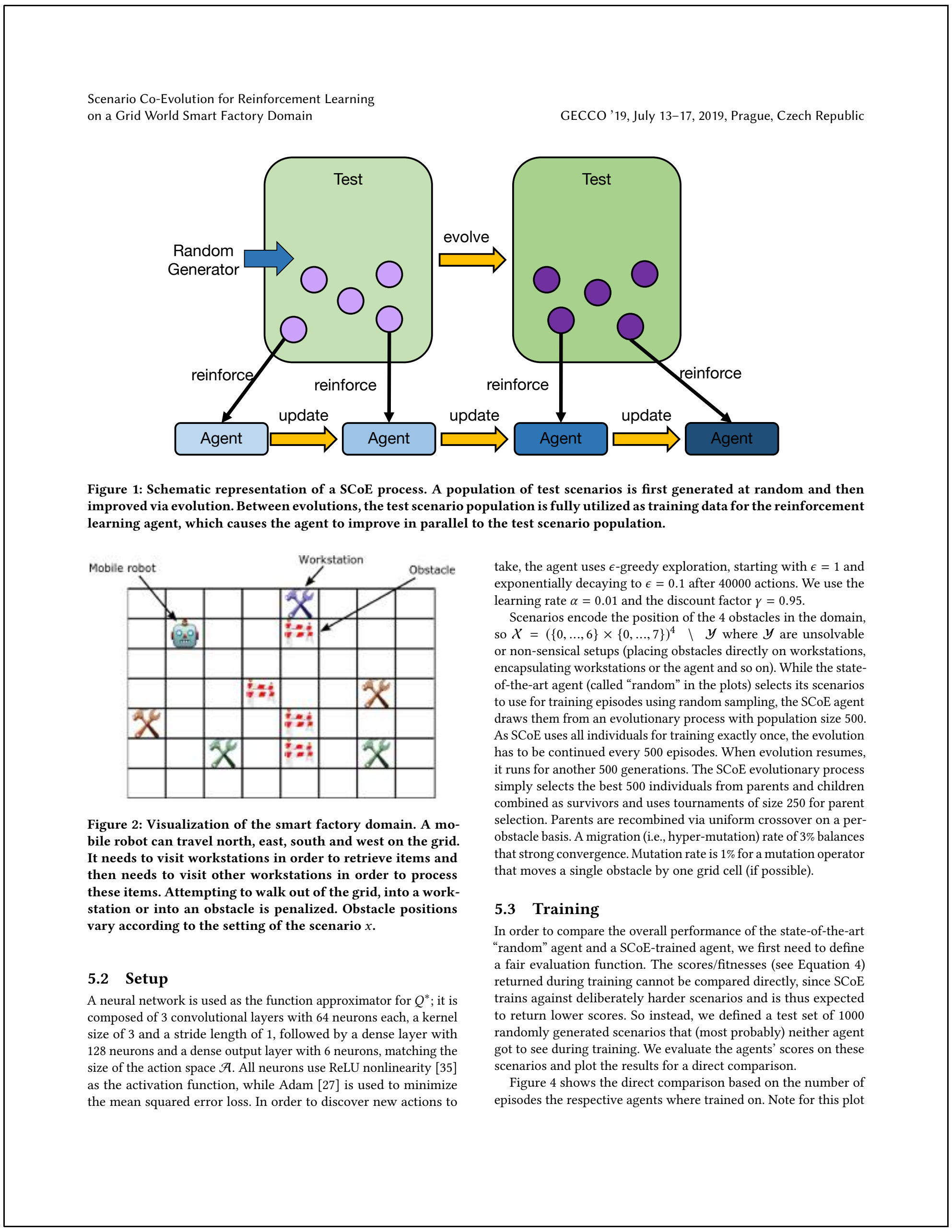
For the evaluation of our approach, we implemented a smart factory domain, in which a number of *items* have to be processed at *workstations* of different types, while avoiding collisions with dynamically placed *obstacles*. Thus, the main focus of the task lies in navigation through the smart factory. However, at certain times the agent also needs to decide which workstation to visit next.

The environment is implemented as a discrete grid of size 7×8 as can be seen in Figure 2. Five workstations of three different types are placed at fixed positions. Five items are placed at these workstations that need to be processed at various other workstations according to an item-specific, fixed sequence of length 1 – 3. While the agent always starts each episode at the fixed position (1, 1), 4 obstacles are placed at varying positions on the grid. The positions of these obstacles are the only free variables in the environment and are either determined randomly, or according to the SCoE method's evolution, i.e., the SCoE approach optimizes for the most impeding position of obstacles to the agent, $\mathcal{X} \subset (\{0, \dots, 6\} \times \{0, \dots, 7\})^4$. However, note that we check for unsolvable instances (when a single workstation is completely blocked, e.g.) and exclude these from both random sampling and evolution.

For the purpose of passing the factory state as an input to the reinforcement learning agent (and in extent its neural network), the factory state is encoded as a stack of 7×8 feature planes, where each plane represents the spatial positions of workstations or the agent w.r.t. to some attribute. See Figure 3 for an informal description of these feature planes.

At each timestep t, the agent can execute a single action a_t from the action space $\mathcal{A}$: move north, south, west, east, pick-up, place. Valid movements, i.e., movements onto free grid fields cause a reward of −1, while collisions with the grid boundary or an obstacle keep the agent's position unchanged and are punished with a reward of −100. A valid pick-up action can only be executed if the agent is not already carrying an item and is standing on a field adjacent to a workstation where an item is available. If the agent is carrying an item, it can execute a valid place action if it is positioned on a field adjacent to a workstation with a type matching the item's next step in the processing sequence. A place action at any other state is considered invalid. The current implementation contains no stochasticity, i.e., the state transitions and rewards are deterministic. A valid pick-up or place action is rewarded with 100, while the reward of an invalid one is −50. An episode is completed if all items were processed correctly.

Taken from original publication: Thomas Gabor, Andreas Sedlmeier, Marie Kiermeier, Thomy Phan, Marcel Henrich, Monika Pichlmair, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, and Jan Wieghardt. Scenario co-evolution for reinforcement learning on a grid world smart factory domain. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 898–906, 2019



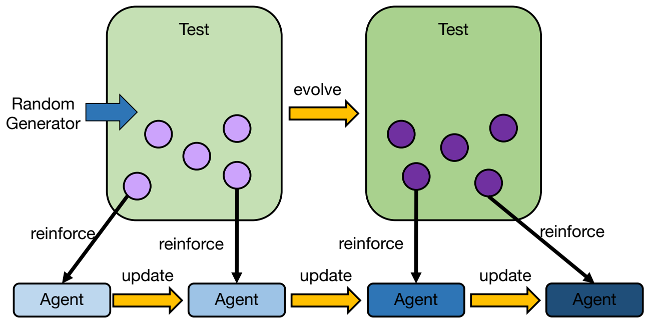


Figure 1: Schematic representation of a SCoE process. A population of test scenarios is first generated at random and then improved via evolution. Between evolutions, the test scenario population is fully utilized as training data for the reinforcement learning agent, which causes the agent to improve in parallel to the test scenario population.

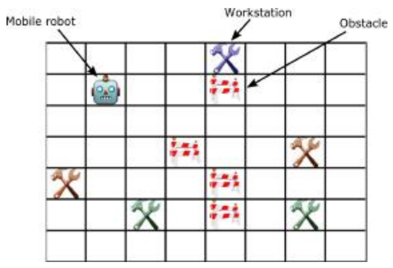


Figure 2: Visualization of the smart factory domain. A mobile robot can travel north, east, south and west on the grid. It needs to visit workstations in order to retrieve items and then needs to visit other workstations in order to process these items. Attempting to walk out of the grid, into a workstation or into an obstacle is penalized. Obstacle positions vary according to the setting of the scenario x.

5.2 Setup

A neural network is used as the function approximator for Q^*; it is composed of 3 convolutional layers with 64 neurons each, a kernel size of 3 and a stride length of 1, followed by a dense layer with 128 neurons and a dense output layer with 6 neurons, matching the size of the action space $\mathcal{A}$. All neurons use ReLU nonlinearity [35] as the activation function, while Adam [27] is used to minimize the mean squared error loss. In order to discover new actions to take, the agent uses ϵ-greedy exploration, starting with $\epsilon = 1$ and exponentially decaying to $\epsilon = 0.1$ after 40000 actions. We use the learning rate $\alpha = 0.01$ and the discount factor $\gamma = 0.95$.

Scenarios encode the position of the 4 obstacles in the domain, so $\mathcal{X} = (\{0, ..., 6\} \times \{0, ..., 7\})^4 \ \setminus \ \mathcal{Y}$ where $\mathcal{Y}$ are unsolvable or non-sensical setups (placing obstacles directly on workstations, encapsulating workstations or the agent and so on). While the state-of-the-art agent (called "random" in the plots) selects its scenarios to use for training episodes using random sampling, the SCoE agent draws them from an evolutionary process with population size 500. As SCoE uses all individuals for training exactly once, the evolution has to be continued every 500 episodes. When evolution resumes, it runs for another 500 generations. The SCoE evolutionary process simply selects the best 500 individuals from parents and children combined as survivors and uses tournaments of size 250 for parent selection. Parents are recombined via uniform crossover on a per-obstacle basis. A migration (i.e., hyper-mutation) rate of 3% balances that strong convergence. Mutation rate is 1% for a mutation operator that moves a single obstacle by one grid cell (if possible).

5.3 Training

In order to compare the overall performance of the state-of-the-art "random" agent and a SCoE-trained agent, we first need to define a fair evaluation function. The scores/fitnesses (see Equation 4) returned during training cannot be compared directly, since SCoE trains against deliberately harder scenarios and is thus expected to return lower scores. So instead, we defined a test set of 1000 randomly generated scenarios that (most probably) neither agent got to see during training. We evaluate the agents' scores on these scenarios and plot the results for a direct comparison.

Figure 4 shows the direct comparison based on the number of episodes the respective agents where trained on. Note for this plot

Taken from original publication: Thomas Gabor, Andreas Sedlmeier, Marie Kiermeier, Thomy Phan, Marcel Henrich, Monika Pichlmair, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, and Jan Wieghardt. Scenario co-evolution for reinforcement learning on a grid world smart factory domain. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 898–906, 2019

Plane	Feature	Description
1	Agent position	The agent's position on the grid
2	Obstacle positions	The positions of the obstacles on the grid
3	Workstation positions	The positions of the workstations on the grid
4	Items for pick-up	The amount of items that can be picked up at the respective positions
5	Item place positions	If the agent carries an item, the positions where this item can be placed at

Figure 3: Description of all feature planes contained in the state input s_t for Q_θ.

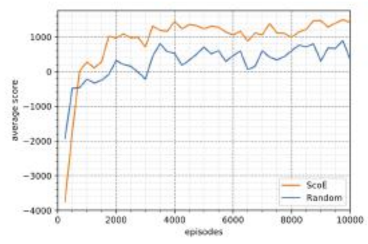


Figure 4: Scores achieved by SCoE and standard "random" reinforcement learning during training over 10000 episodes. Scores are averages of running the current agent against 1000 randomly generated test scenarios.

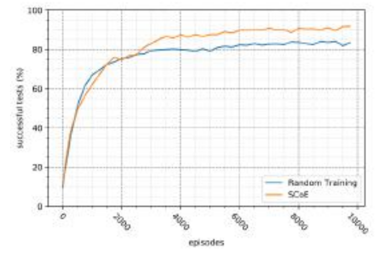


Figure 6: Percentage of successfully solved test scenarios by SCoE and standard "random" reinforcement learning. The values are calculated from a randomly generated set of 1000 scenarios.

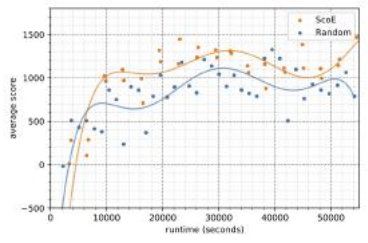


Figure 5: Scores achieved by a SCoE and standard "random" reinforcement learning during training for ≈ 50000 seconds of runtime. Scores are averages of running the current agent against 1000 randomly generated test scenarios. The plot shows single runs with an added trend line. Over the same amount of training time, SCoE generally achieves slighty higher average scores.

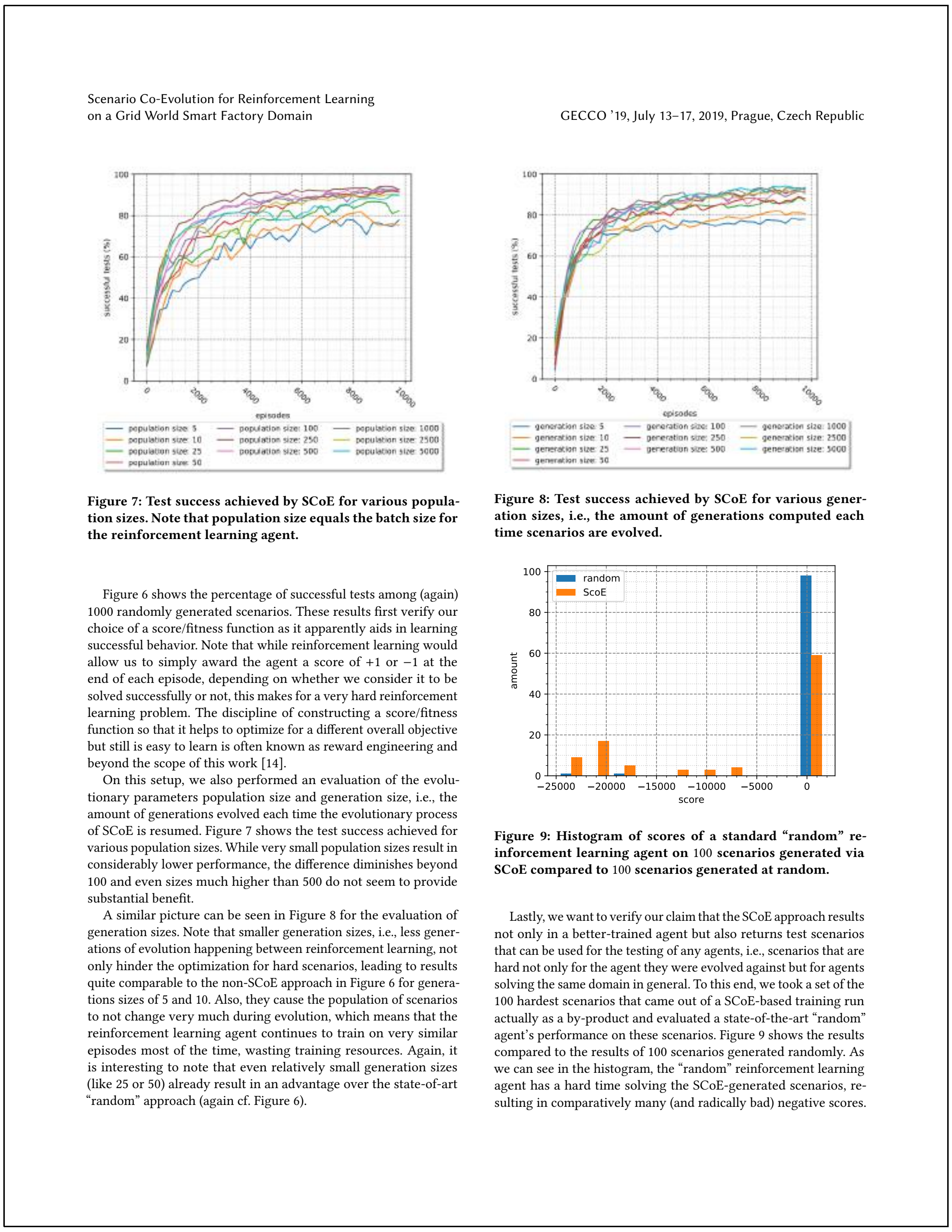
we took a snapshot of the agent every 250 episodes, resulting in the horizontal resolution of the plot. While we can see clearly that SCoE outperforms the "random" agent even on randomly generated scenarios, it does have a bit of an unfair advantage: Sampling scenarios randomly obviously takes less computational effort than running several hundred generations of evolution to get the hardest scenarios.

For this reason, we plotted the same data according to runtime in Figure 5 as measured in physical seconds running on a standard computer. The trend line still shows a net benefit of using SCoE with respect to the time-quality trade-off. This means that implementing SCoE and running the elaborate evolutionary process in contrast to just using random sampling for training scenarios actually pays off in performance.

5.4 Test

As stated in the introduction, improving scores is of course a nice benefit, but especially in real-world applications we are often more interested in the agent avoiding complete failures rather than getting the last bits of performance in already good scenarios. The smart factory domain was constructed in such a way that it has a clear overall goal: A sequence of actions is successful iff in the end all items have been fully processed, i.e., have been transported to all the workstations they needed to visit.

Taken from original publication: Thomas Gabor, Andreas Sedlmeier, Marie Kiermeier, Thomy Phan, Marcel Henrich, Monika Pichlmair, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, and Jan Wieghardt. Scenario co-evolution for reinforcement learning on a grid world smart factory domain. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 898–906, 2019



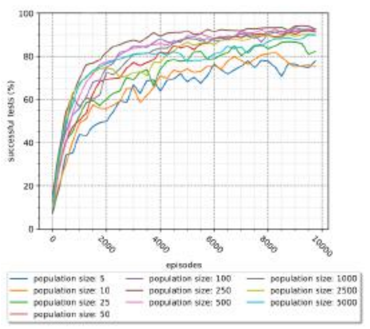


Figure 7: Test success achieved by SCoE for various population sizes. Note that population size equals the batch size for the reinforcement learning agent.

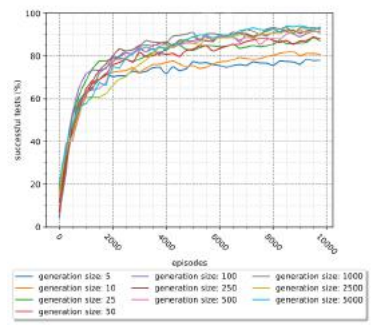


Figure 8: Test success achieved by SCoE for various generation sizes, i.e., the amount of generations computed each time scenarios are evolved.

Figure 6 shows the percentage of successful tests among (again) 1000 randomly generated scenarios. These results first verify our choice of a score/fitness function as it apparently aids in learning successful behavior. Note that while reinforcement learning would allow us to simply award the agent a score of +1 or −1 at the end of each episode, depending on whether we consider it to be solved successfully or not, this makes for a very hard reinforcement learning problem. The discipline of constructing a score/fitness function so that it helps to optimize for a different overall objective but still is easy to learn is often known as reward engineering and beyond the scope of this work [14].

On this setup, we also performed an evaluation of the evolutionary parameters population size and generation size, i.e., the amount of generations evolved each time the evolutionary process of SCoE is resumed. Figure 7 shows the test success achieved for various population sizes. While very small population sizes result in considerably lower performance, the difference diminishes beyond 100 and even sizes much higher than 500 do not seem to provide substantial benefit.

A similar picture can be seen in Figure 8 for the evaluation of generation sizes. Note that smaller generation sizes, i.e., less generations of evolution happening between reinforcement learning, not only hinder the optimization for hard scenarios, leading to results quite comparable to the non-SCoE approach in Figure 6 for generations sizes of 5 and 10. Also, they cause the population of scenarios to not change very much during evolution, which means that the reinforcement learning agent continues to train on very similar episodes most of the time, wasting training resources. Again, it is interesting to note that even relatively small generation sizes (like 25 or 50) already result in an advantage over the state-of-art "random" approach (again cf. Figure 6).

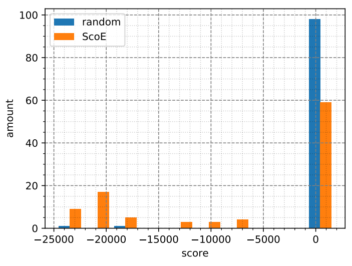
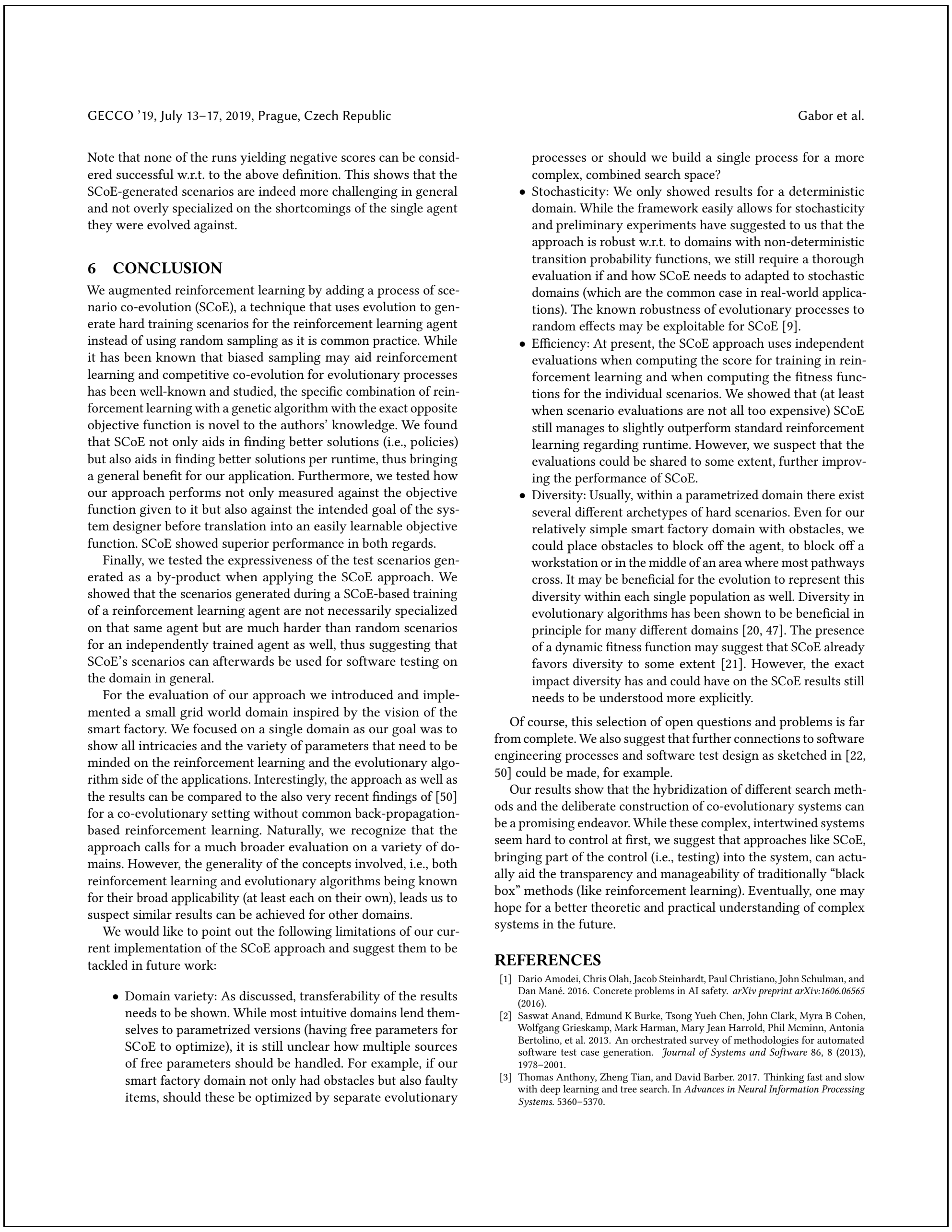


Figure 9: Histogram of scores of a standard "random" reinforcement learning agent on 100 scenarios generated via SCoE compared to 100 scenarios generated at random.

Lastly, we want to verify our claim that the SCoE approach results not only in a better-trained agent but also returns test scenarios that can be used for the testing of any agents, i.e., scenarios that are hard not only for the agent they were evolved against but for agents solving the same domain in general. To this end, we took a set of the 100 hardest scenarios that came out of a SCoE-based training run actually as a by-product and evaluated a state-of-the-art "random" agent's performance on these scenarios. Figure 9 shows the results compared to the results of 100 scenarios generated randomly. As we can see in the histogram, the "random" reinforcement learning agent has a hard time solving the SCoE-generated scenarios, resulting in comparatively many (and radically bad) negative scores.

Taken from original publication: Thomas Gabor, Andreas Sedlmeier, Marie Kiermeier, Thomy Phan, Marcel Henrich, Monika Pichlmair, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, and Jan Wieghardt. Scenario co-evolution for reinforcement learning on a grid world smart factory domain. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 898–906, 2019



Note that none of the runs yielding negative scores can be considered successful w.r.t. to the above definition. This shows that the SCoE-generated scenarios are indeed more challenging in general and not overly specialized on the shortcomings of the single agent they were evolved against.

6 CONCLUSION

We augmented reinforcement learning by adding a process of scenario co-evolution (SCoE), a technique that uses evolution to generate hard training scenarios for the reinforcement learning agent instead of using random sampling as it is common practice. While it has been known that biased sampling may aid reinforcement learning and competitive co-evolution for evolutionary processes has been well-known and studied, the specific combination of reinforcement learning with a genetic algorithm with the exact opposite objective function is novel to the authors' knowledge. We found that SCoE not only aids in finding better solutions (i.e., policies) but also aids in finding better solutions per runtime, thus bringing a general benefit for our application. Furthermore, we tested how our approach performs not only measured against the objective function given to it but also against the intended goal of the system designer before translation into an easily learnable objective function. SCoE showed superior performance in both regards.

Finally, we tested the expressiveness of the test scenarios generated as a by-product when applying the SCoE approach. We showed that the scenarios generated during a SCoE-based training of a reinforcement learning agent are not necessarily specialized on that same agent but are much harder than random scenarios for an independently trained agent as well, thus suggesting that SCoE's scenarios can afterwards be used for software testing on the domain in general.

For the evaluation of our approach we introduced and implemented a small grid world domain inspired by the vision of the smart factory. We focused on a single domain as our goal was to show all intricacies and the variety of parameters that need to be minded on the reinforcement learning and the evolutionary algorithm side of the applications. Interestingly, the approach as well as the results can be compared to the also very recent findings of [50] for a co-evolutionary setting without common back-propagation-based reinforcement learning. Naturally, we recognize that the approach calls for a much broader evaluation on a variety of domains. However, the generality of the concepts involved, i.e., both reinforcement learning and evolutionary algorithms being known for their broad applicability (at least each on their own), leads us to suspect similar results can be achieved for other domains.

We would like to point out the following limitations of our current implementation of the SCoE approach and suggest them to be tackled in future work:

- Domain variety: As discussed, transferability of the results needs to be shown. While most intuitive domains lend themselves to parametrized versions (having free parameters for SCoE to optimize), it is still unclear how multiple sources of free parameters should be handled. For example, if our smart factory domain not only had obstacles but also faulty items, should these be optimized by separate evolutionary processes or should we build a single process for a more complex, combined search space?
- Stochasticity: We only showed results for a deterministic domain. While the framework easily allows for stochasticity and preliminary experiments have suggested to us that the approach is robust w.r.t. to domains with non-deterministic transition probability functions, we still require a thorough evaluation if and how SCoE needs to adapted to stochastic domains (which are the common case in real-world applications). The known robustness of evolutionary processes to random effects may be exploitable for SCoE [9].
- Efficiency: At present, the SCoE approach uses independent evaluations when computing the score for training in reinforcement learning and when computing the fitness functions for the individual scenarios. We showed that (at least when scenario evaluations are not all too expensive) SCoE still manages to slightly outperform standard reinforcement learning regarding runtime. However, we suspect that the evaluations could be shared to some extent, further improving the performance of SCoE.
- Diversity: Usually, within a parametrized domain there exist several different archetypes of hard scenarios. Even for our relatively simple smart factory domain with obstacles, we could place obstacles to block off the agent, to block off a workstation or in the middle of an area where most pathways cross. It may be beneficial for the evolution to represent this diversity within each single population as well. Diversity in evolutionary algorithms has been shown to be beneficial in principle for many different domains [20, 47]. The presence of a dynamic fitness function may suggest that SCoE already favors diversity to some extent [21]. However, the exact impact diversity has and could have on the SCoE results still needs to be understood more explicitly.

Of course, this selection of open questions and problems is far from complete. We also suggest that further connections to software engineering processes and software test design as sketched in [22, 50] could be made, for example.

Our results show that the hybridization of different search methods and the deliberate construction of co-evolutionary systems can be a promising endeavor. While these complex, intertwined systems seem hard to control at first, we suggest that approaches like SCoE, bringing part of the control (i.e., testing) into the system, can actually aid the transparency and manageability of traditionally "black box" methods (like reinforcement learning). Eventually, one may hope for a better theoretic and practical understanding of complex systems in the future.

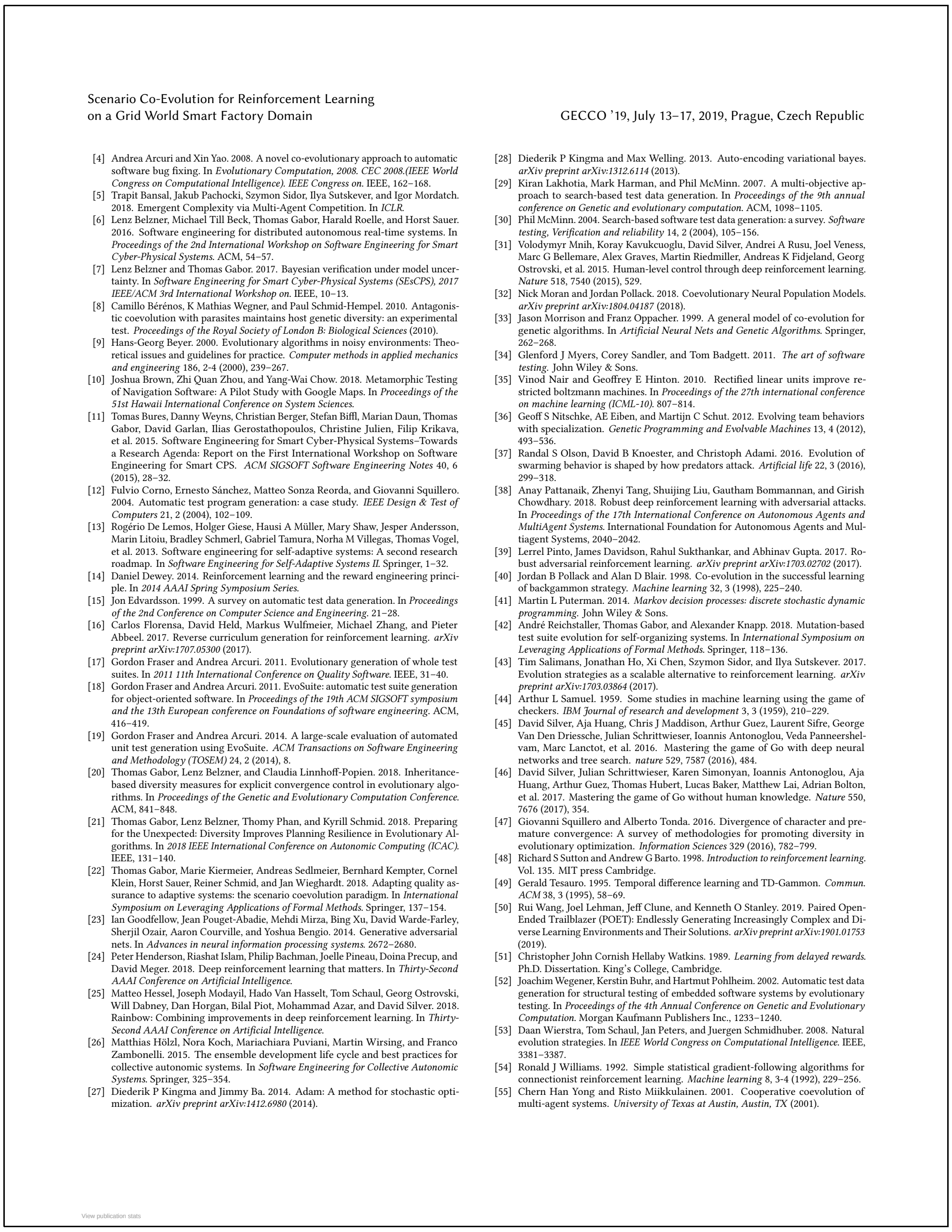
REFERENCES

[1] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. 2016. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565* (2016).

[2] Saswat Anand, Edmund K Burke, Tsong Yueh Chen, John Clark, Myra B Cohen, Wolfgang Grieskamp, Mark Harman, Mary Jean Harrold, Phil Mcminn, Antonia Bertolino, et al. 2013. An orchestrated survey of methodologies for automated software test case generation. *Journal of Systems and Software* 86, 8 (2013), 1978–2001.

[3] Thomas Anthony, Zheng Tian, and David Barber. 2017. Thinking fast and slow with deep learning and tree search. In *Advances in Neural Information Processing Systems*. 5360–5370.

Taken from original publication: Thomas Gabor, Andreas Sedlmeier, Marie Kiermeier, Thomy Phan, Marcel Henrich, Monika Pichlmair, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, and Jan Wieghardt. Scenario co-evolution for reinforcement learning on a grid world smart factory domain. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 898–906, 2019



[4] Andrea Arcuri and Xin Yao. 2008. A novel co-evolutionary approach to automatic software bug fixing. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*. IEEE, 162–168.

[5] Trapit Bansal, Jakub Pachocki, Szymon Sidor, Ilya Sutskever, and Igor Mordatch. 2018. Emergent Complexity via Multi-Agent Competition. In *ICLR*.

[6] Lenz Belzner, Michael Till Beck, Thomas Gabor, Harald Roelle, and Horst Sauer. 2016. Software engineering for distributed autonomous real-time systems. In *Proceedings of the 2nd International Workshop on Software Engineering for Smart Cyber-Physical Systems*. ACM, 54–57.

[7] Lenz Belzner and Thomas Gabor. 2017. Bayesian verification under model uncertainty. In *Software Engineering for Smart Cyber-Physical Systems (SEsCPS), 2017 IEEE/ACM 3rd International Workshop on*. IEEE, 10–13.

[8] Camillo Bérénos, K Mathias Wegner, and Paul Schmid-Hempel. 2010. Antagonistic coevolution with parasites maintains host genetic diversity: an experimental test. *Proceedings of the Royal Society of London B: Biological Sciences* (2010).

[9] Hans-Georg Beyer. 2000. Evolutionary algorithms in noisy environments: Theoretical issues and guidelines for practice. *Computer methods in applied mechanics and engineering* 186, 2-4 (2000), 239–267.

[10] Joshua Brown, Zhi Quan Zhou, and Yang-Wai Chow. 2018. Metamorphic Testing of Navigation Software: A Pilot Study with Google Maps. In *Proceedings of the 51st Hawaii International Conference on System Sciences*.

[11] Tomas Bures, Danny Weyns, Christian Berger, Stefan Biffl, Marian Daun, Thomas Gabor, David Garlan, Ilias Gerostathopoulos, Christine Julien, Filip Krikava, et al. 2015. Software Engineering for Smart Cyber-Physical Systems–Towards a Research Agenda: Report on the First International Workshop on Software Engineering for Smart CPS. *ACM SIGSOFT Software Engineering Notes* 40, 6 (2015), 28–32.

[12] Fulvio Corno, Ernesto Sánchez, Matteo Sonza Reorda, and Giovanni Squillero. 2004. Automatic test program generation: a case study. *IEEE Design & Test of Computers* 21, 2 (2004), 102–109.

[13] Rogério De Lemos, Holger Giese, Hausi A Müller, Mary Shaw, Jesper Andersson, Marin Litoiu, Bradley Schmerl, Gabriel Tamura, Norha M Villegas, Thomas Vogel, et al. 2013. Software engineering for self-adaptive systems: A second research roadmap. In *Software Engineering for Self-Adaptive Systems II*. Springer, 1–32.

[14] Daniel Dewey. 2014. Reinforcement learning and the reward engineering principle. In *2014 AAAI Spring Symposium Series*.

[15] Jon Edvardsson. 1999. A survey on automatic test data generation. In *Proceedings of the 2nd Conference on Computer Science and Engineering*. 21–28.

[16] Carlos Florensa, David Held, Markus Wulfmeier, Michael Zhang, and Pieter Abbeel. 2017. Reverse curriculum generation for reinforcement learning. *arXiv preprint arXiv:1707.05300* (2017).

[17] Gordon Fraser and Andrea Arcuri. 2011. Evolutionary generation of whole test suites. In *2011 11th International Conference on Quality Software*. IEEE, 31–40.

[18] Gordon Fraser and Andrea Arcuri. 2011. EvoSuite: automatic test suite generation for object-oriented software. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. ACM, 416–419.

[19] Gordon Fraser and Andrea Arcuri. 2014. A large-scale evaluation of automated unit test generation using EvoSuite. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 24, 2 (2014), 8.

[20] Thomas Gabor, Lenz Belzner, and Claudia Linnhoff-Popien. 2018. Inheritance-based diversity measures for explicit convergence control in evolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 841–848.

[21] Thomas Gabor, Lenz Belzner, Thomy Phan, and Kyrill Schmid. 2018. Preparing for the Unexpected: Diversity Improves Planning Resilience in Evolutionary Algorithms. In *2018 IEEE International Conference on Autonomic Computing (ICAC)*. IEEE, 131–140.

[22] Thomas Gabor, Marie Kiermeier, Andreas Sedlmeier, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, and Jan Wieghardt. 2018. Adapting quality assurance to adaptive systems: the scenario coevolution paradigm. In *International Symposium on Leveraging Applications of Formal Methods*. Springer, 137–154.

[23] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*. 2672–2680.

[24] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. 2018. Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[25] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. 2018. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[26] Matthias Hölzl, Nora Koch, Mariachiara Puviani, Martin Wirsing, and Franco Zambonelli. 2015. The ensemble development life cycle and best practices for collective autonomic systems. In *Software Engineering for Collective Autonomic Systems*. Springer, 325–354.

[27] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[28] Diederik P Kingma and Max Welling. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114* (2013).

[29] Kiran Lakhotia, Mark Harman, and Phil McMinn. 2007. A multi-objective approach to search-based test data generation. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*. ACM, 1098–1105.

[30] Phil McMinn. 2004. Search-based software test data generation: a survey. *Software testing, Verification and reliability* 14, 2 (2004), 105–156.

[31] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529.

[32] Nick Moran and Jordan Pollack. 2018. Coevolutionary Neural Population Models. *arXiv preprint arXiv:1804.04187* (2018).

[33] Jason Morrison and Franz Oppacher. 1999. A general model of co-evolution for genetic algorithms. In *Artificial Neural Nets and Genetic Algorithms*. Springer, 262–268.

[34] Glenford J Myers, Corey Sandler, and Tom Badgett. 2011. *The art of software testing*. John Wiley & Sons.

[35] Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*. 807–814.

[36] Geoff S Nitschke, AE Eiben, and Martijn C Schut. 2012. Evolving team behaviors with specialization. *Genetic Programming and Evolvable Machines* 13, 4 (2012), 493–536.

[37] Randal S Olson, David B Knoester, and Christoph Adami. 2016. Evolution of swarming behavior is shaped by how predators attack. *Artificial life* 22, 3 (2016), 299–318.

[38] Anay Pattanaik, Zhenyi Tang, Shuijing Liu, Gautham Bommannan, and Girish Chowdhary. 2018. Robust deep reinforcement learning with adversarial attacks. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2040–2042.

[39] Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. 2017. Robust adversarial reinforcement learning. *arXiv preprint arXiv:1703.02702* (2017).

[40] Jordan B Pollack and Alan D Blair. 1998. Co-evolution in the successful learning of backgammon strategy. *Machine learning* 32, 3 (1998), 225–240.

[41] Martin L Puterman. 2014. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

[42] André Reichstaller, Thomas Gabor, and Alexander Knapp. 2018. Mutation-based test suite evolution for self-organizing systems. In *International Symposium on Leveraging Applications of Formal Methods*. Springer, 118–136.

[43] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. 2017. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864* (2017).

[44] Arthur L Samuel. 1959. Some studies in machine learning using the game of checkers. *IBM Journal of research and development* 3, 3 (1959), 210–229.

[45] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016), 484.

[46] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of Go without human knowledge. *Nature* 550, 7676 (2017), 354.

[47] Giovanni Squillero and Alberto Tonda. 2016. Divergence of character and premature convergence: A survey of methodologies for promoting diversity in evolutionary optimization. *Information Sciences* 329 (2016), 782–799.

[48] Richard S Sutton and Andrew G Barto. 1998. *Introduction to reinforcement learning*. Vol. 135. MIT press Cambridge.

[49] Gerald Tesauro. 1995. Temporal difference learning and TD-Gammon. *Commun. ACM* 38, 3 (1995), 58–69.

[50] Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O Stanley. 2019. Paired Open-Ended Trailblazer (POET): Endlessly Generating Increasingly Complex and Diverse Learning Environments and Their Solutions. *arXiv preprint arXiv:1901.01753* (2019).

[51] Christopher John Cornish Hellaby Watkins. 1989. *Learning from delayed rewards*. Ph.D. Dissertation. King's College, Cambridge.

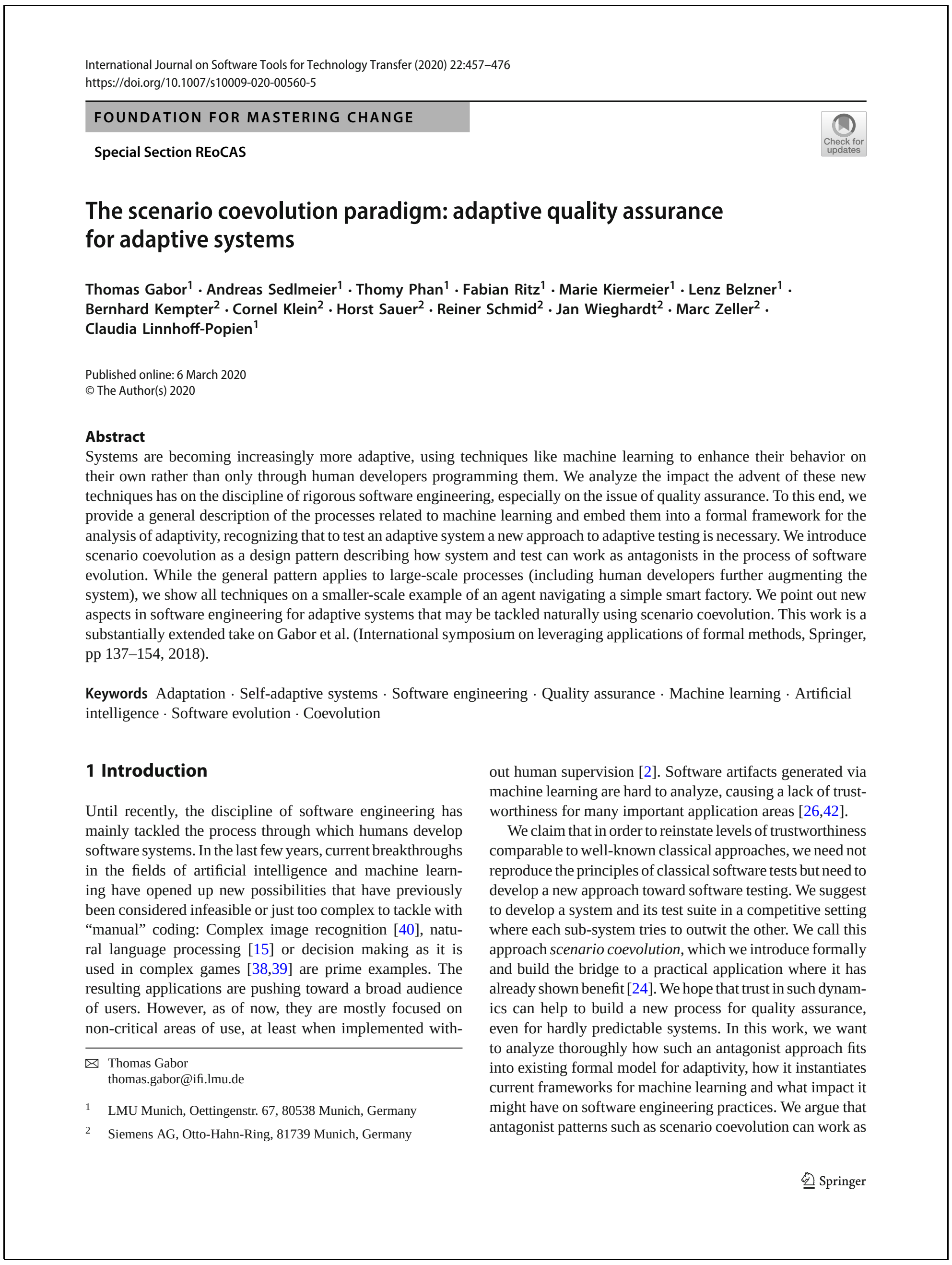
[52] Joachim Wegener, Kerstin Buhr, and Hartmut Pohlheim. 2002. Automatic test data generation for structural testing of embedded software systems by evolutionary testing. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*. Morgan Kaufmann Publishers Inc., 1233–1240.

[53] Daan Wierstra, Tom Schaul, Jan Peters, and Juergen Schmidhuber. 2008. Natural evolution strategies. In *IEEE World Congress on Computational Intelligence*. IEEE, 3381–3387.

[54] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8, 3-4 (1992), 229–256.

[55] Chern Han Yong and Risto Miikkulainen. 2001. Cooperative coevolution of multi-agent systems. *University of Texas at Austin, Austin, TX* (2001).

Taken from original publication: Thomas Gabor, Andreas Sedlmeier, Thomy Phan, Fabian Ritz, Marie Kiermeier, Lenz Belzner, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, Marc Zeller, and Claudia Linnhoff-Popien. The scenario coevolution paradigm: adaptive quality assurance for adaptive systems. *International Journal on Software Tools for Technology Transfer*, pages 1–20, 2020



International Journal on Software Tools for Technology Transfer (2020) 22:457–476
https://doi.org/10.1007/s10009-020-00560-5

FOUNDATION FOR MASTERING CHANGE

Special Section REoCAS

The scenario coevolution paradigm: adaptive quality assurance for adaptive systems

Thomas Gabor[1] · Andreas Sedlmeier[1] · Thomy Phan[1] · Fabian Ritz[1] · Marie Kiermeier[1] · Lenz Belzner[1] · Bernhard Kempter[2] · Cornel Klein[2] · Horst Sauer[2] · Reiner Schmid[2] · Jan Wieghardt[2] · Marc Zeller[2] · Claudia Linnhoff-Popien[1]

Published online: 6 March 2020

Abstract
Systems are becoming increasingly more adaptive, using techniques like machine learning to enhance their behavior on their own rather than only through human developers programming them. We analyze the impact the advent of these new techniques has on the discipline of rigorous software engineering, especially on the issue of quality assurance. To this end, we provide a general description of the processes related to machine learning and embed them into a formal framework for the analysis of adaptivity, recognizing that to test an adaptive system a new approach to adaptive testing is necessary. We introduce scenario coevolution as a design pattern describing how system and test can work as antagonists in the process of software evolution. While the general pattern applies to large-scale processes (including human developers further augmenting the system), we show all techniques on a smaller-scale example of an agent navigating a simple smart factory. We point out new aspects in software engineering for adaptive systems that may be tackled naturally using scenario coevolution. This work is a substantially extended take on Gabor et al. (International symposium on leveraging applications of formal methods, Springer, pp 137–154, 2018).

Keywords Adaptation · Self-adaptive systems · Software engineering · Quality assurance · Machine learning · Artificial intelligence · Software evolution · Coevolution

1 Introduction

Until recently, the discipline of software engineering has mainly tackled the process through which humans develop software systems. In the last few years, current breakthroughs in the fields of artificial intelligence and machine learning have opened up new possibilities that have previously been considered infeasible or just too complex to tackle with "manual" coding: Complex image recognition [40], natural language processing [15] or decision making as it is used in complex games [38,39] are prime examples. The resulting applications are pushing toward a broad audience of users. However, as of now, they are mostly focused on non-critical areas of use, at least when implemented without human supervision [2]. Software artifacts generated via machine learning are hard to analyze, causing a lack of trustworthiness for many important application areas [26,42].

We claim that in order to reinstate levels of trustworthiness comparable to well-known classical approaches, we need not reproduce the principles of classical software tests but need to develop a new approach toward software testing. We suggest to develop a system and its test suite in a competitive setting where each sub-system tries to outwit the other. We call this approach *scenario coevolution*, which we introduce formally and build the bridge to a practical application where it has already shown benefit [24]. We hope that trust in such dynamics can help to build a new process for quality assurance, even for hardly predictable systems. In this work, we want to analyze thoroughly how such an antagonist approach fits into existing formal model for adaptivity, how it instantiates current frameworks for machine learning and what impact it might have on software engineering practices. We argue that antagonist patterns such as scenario coevolution can work as

✉ Thomas Gabor
thomas.gabor@ifi.lmu.de

1 LMU Munich, Oettingenstr. 67, 80538 Munich, Germany
2 Siemens AG, Otto-Hahn-Ring, 81739 Munich, Germany

Taken from original publication: Thomas Gabor, Andreas Sedlmeier, Thomy Phan, Fabian Ritz, Marie Kiermeier, Lenz Belzner, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, Marc Zeller, and Claudia Linnhoff-Popien. The scenario coevolution paradigm: adaptive quality assurance for adaptive systems. *International Journal on Software Tools for Technology Transfer*, pages 1–20, 2020

a unifying concept across all these domains and eventually enable more powerful adaptive quality assurance.

In this paper, we substantially expand the work on this topic presented in [23]. Section 2 provides a short overview on related work on process models for the development of adaptive software. Following a top-down approach, we start with the description of our approach in Sect. 3 by extending a formal framework for the description of systems first introduced in [28] and augment it to also include the process of software and system development. We use said framework to first present a formal definition of an example domain used in [24] in Sect. 4. Section 5 discusses state-of-the-art algorithms to achieve adaptation and introduces the machine learning pipeline, a process model specifically designed to engineer machine learning components. From this, we derive four core concepts for the engineering of adaptive systems in Sect. 6. In order to integrate these with our formal framework, Sect. 7 introduces our notion of scenarios and their application to an incremental software testing process. In Sect. 8, we apply this new notion to our example domain, formally explaining the results of [24]. In Sect. 9, we discuss which effects scenario coevolution has on a selection of practical software engineering tasks and how it helps implement the core concepts. Finally, Sect. 10 provides a brief conclusion.

2 Related work

Many researchers and practitioners in recent years have already been concerned about the changes necessary to allow for solid and reliable software engineering processes for (self-)adaptive systems. Central challenges were collected in [36], where issues of quality assurance are already mentioned but the focus is more on bringing about complex adaptive behavior in the first place. The later research roadmap of [17] puts a strong focus on interaction patterns of already adaptive systems (both between each other and with human developers) and already dedicates a section to verification and validation issues, being close in mind to the perspective of this work. We fall in line with the roadmap further specified in [7,12,13].

While this work largely builds upon [28], there have been other approaches to formalize the notion of adaptivity: [34] discusses high-level architectural patterns that form multiple interconnected adaptation loops. In [4], such feedback loops are based on the MAPE-K model [29]. While these approaches largely focus on the formal construction of adaptive systems, there have also been approaches that assume a (more human-centric or at least tool-centric) software engineering perspective [3,19,22,45]. We want to discuss two of those on greater detail.

In the results of the *ASCENS* (Autonomous Service Component ENSembles) project [46], the interplay between human developers and autonomous adaptation has been formalized in a life cycle model featuring separate states for each the development progress of each respective feedback cycle. Classical software development tasks and self-adaptation (as well as self-monitoring and self-awareness) are regarded as equally powerful contributing mechanisms for the production of software. Both can be employed in junction to steer the development process. In addition, ASCENS built upon a (in parts) similar formal notion of adaptivity [11,32] and sketched a connection between adaptivity in complex distributed systems and multi-goal multi-agent learning [27].

ADELFE (Atelier de Développement de Logiciels à Fonctionnalité Emergente) is a toolkit designed to augment current development processes to account for complex adaptive systems [8,9]. For this purpose, the ADELFE process is based on the Rational Unified Process (RUP) [31] and comes with tools for various tasks of software design. From a more scientific point of view, ADELFE is also based on the theory of adaptive multi-agent systems. For ADELFE, multi-agent systems are used to derive a set of stereotypes for components, which ease modeling for according types of systems. It thus imposes stronger restrictions on system design than our approach intends to.

Besides the field of software engineering, the field of artificial intelligence research is currently (re-)discovering a lot of the same issues the discipline of engineering for complex adaptive systems faced: The highly complex and opaque nature of machine learning algorithms and the resulting data structures often forces black-box testing and makes possible guarantees weak. When online learning is employed, the algorithm's behavior is subject to great variance and testing usually needs to work online as well. The seminal paper [2] provides a good overview of the issues. When applying artificial intelligence to a large variety of products, rigorous engineering for this kind of software seems to be one of the major necessities lacking at the moment.

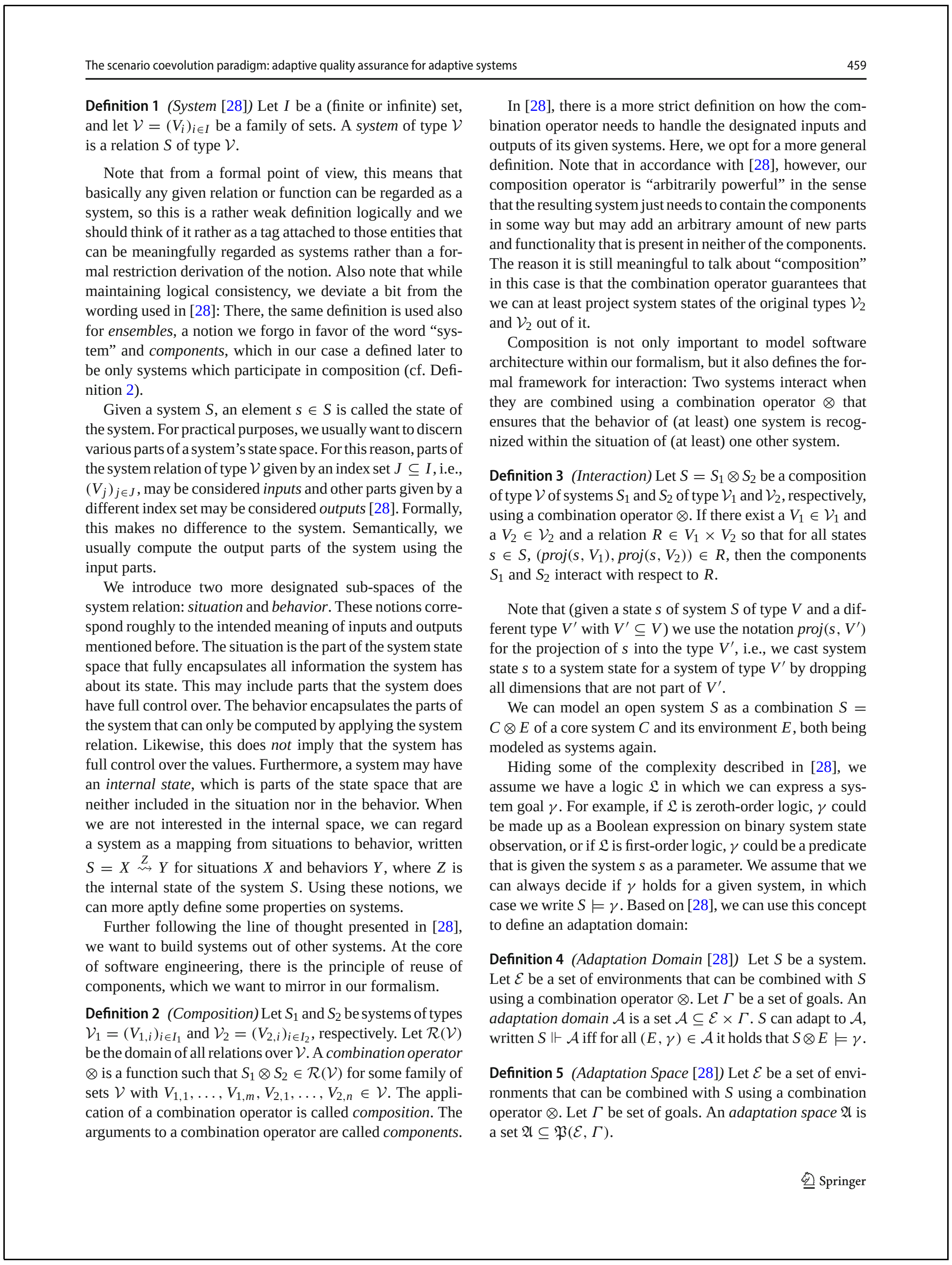
3 Formal framework

In this section, we introduce a formal framework as a basis for our analysis. We first build upon the framework described in [28] to define adaptive systems and then proceed to reason about the influence of their inherent structure on software architecture. In the last subsection, we introduce an example system and realize the formal definitions in its context.

3.1 Describing adaptive systems

We roughly adopt the formal definitions of our vocabulary related to the description of systems from [28]: We describe a system as an arbitrary relation over any given set of variables.

Taken from original publication: Thomas Gabor, Andreas Sedlmeier, Thomy Phan, Fabian Ritz, Marie Kiermeier, Lenz Belzner, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, Marc Zeller, and Claudia Linnhoff-Popien. The scenario coevolution paradigm: adaptive quality assurance for adaptive systems. *International Journal on Software Tools for Technology Transfer*, pages 1–20, 2020



Definition 1 *(System* [28]*)* Let I be a (finite or infinite) set, and let $\mathcal{V} = (V_i)_{i \in I}$ be a family of sets. A *system* of type $\mathcal{V}$ is a relation S of type $\mathcal{V}$.

Note that from a formal point of view, this means that basically any given relation or function can be regarded as a system, so this is a rather weak definition logically and we should think of it rather as a tag attached to those entities that can be meaningfully regarded as systems rather than a formal restriction derivation of the notion. Also note that while maintaining logical consistency, we deviate a bit from the wording used in [28]: There, the same definition is used also for *ensembles*, a notion we forgo in favor of the word "system" and *components*, which in our case a defined later to be only systems which participate in composition (cf. Definition 2).

Given a system S, an element $s \in S$ is called the state of the system. For practical purposes, we usually want to discern various parts of a system's state space. For this reason, parts of the system relation of type $\mathcal{V}$ given by an index set $J \subseteq I$, i.e., $(V_j)_{j \in J}$, may be considered *inputs* and other parts given by a different index set may be considered *outputs* [28]. Formally, this makes no difference to the system. Semantically, we usually compute the output parts of the system using the input parts.

We introduce two more designated sub-spaces of the system relation: *situation* and *behavior*. These notions correspond roughly to the intended meaning of inputs and outputs mentioned before. The situation is the part of the system state space that fully encapsulates all information the system has about its state. This may include parts that the system does have full control over. The behavior encapsulates the parts of the system that can only be computed by applying the system relation. Likewise, this does *not* imply that the system has full control over the values. Furthermore, a system may have an *internal state*, which is parts of the state space that are neither included in the situation nor in the behavior. When we are not interested in the internal space, we can regard a system as a mapping from situations to behavior, written $S = X \overset{Z}{\rightsquigarrow} Y$ for situations X and behaviors Y, where Z is the internal state of the system S. Using these notions, we can more aptly define some properties on systems.

Further following the line of thought presented in [28], we want to build systems out of other systems. At the core of software engineering, there is the principle of reuse of components, which we want to mirror in our formalism.

Definition 2 *(Composition)* Let S_1 and S_2 be systems of types $\mathcal{V}_1 = (V_{1,i})_{i \in I_1}$ and $\mathcal{V}_2 = (V_{2,i})_{i \in I_2}$, respectively. Let $\mathcal{R}(\mathcal{V})$ be the domain of all relations over $\mathcal{V}$. A *combination operator* $\otimes$ is a function such that $S_1 \otimes S_2 \in \mathcal{R}(\mathcal{V})$ for some family of sets $\mathcal{V}$ with $V_{1,1}, \dots, V_{1,m}, V_{2,1}, \dots, V_{2,n} \in \mathcal{V}$. The application of a combination operator is called *composition*. The arguments to a combination operator are called *components*.

In [28], there is a more strict definition on how the combination operator needs to handle the designated inputs and outputs of its given systems. Here, we opt for a more general definition. Note that in accordance with [28], however, our composition operator is "arbitrarily powerful" in the sense that the resulting system just needs to contain the components in some way but may add an arbitrary amount of new parts and functionality that is present in neither of the components. The reason it is still meaningful to talk about "composition" in this case is that the combination operator guarantees that we can at least project system states of the original types $\mathcal{V}_2$ and $\mathcal{V}_2$ out of it.

Composition is not only important to model software architecture within our formalism, but it also defines the formal framework for interaction: Two systems interact when they are combined using a combination operator $\otimes$ that ensures that the behavior of (at least) one system is recognized within the situation of (at least) one other system.

Definition 3 *(Interaction)* Let $S = S_1 \otimes S_2$ be a composition of type $\mathcal{V}$ of systems S_1 and S_2 of type $\mathcal{V}_1$ and $\mathcal{V}_2$, respectively, using a combination operator $\otimes$. If there exist a $V_1 \in \mathcal{V}_1$ and a $V_2 \in \mathcal{V}_2$ and a relation $R \in V_1 \times V_2$ so that for all states $s \in S$, $(proj(s, V_1), proj(s, V_2)) \in R$, then the components S_1 and S_2 interact with respect to R.

Note that (given a state s of system S of type V and a different type V' with $V' \subseteq V$) we use the notation $proj(s, V')$ for the projection of s into the type V', i.e., we cast system state s to a system state for a system of type V' by dropping all dimensions that are not part of V'.

We can model an open system S as a combination $S = C \otimes E$ of a core system C and its environment E, both being modeled as systems again.

Hiding some of the complexity described in [28], we assume we have a logic $\mathfrak{L}$ in which we can express a system goal γ. For example, if $\mathfrak{L}$ is zeroth-order logic, γ could be made up as a Boolean expression on binary system state observation, or if $\mathfrak{L}$ is first-order logic, γ could be a predicate that is given the system s as a parameter. We assume that we can always decide if γ holds for a given system, in which case we write $S \models \gamma$. Based on [28], we can use this concept to define an adaptation domain:

Definition 4 *(Adaptation Domain* [28]*)* Let S be a system. Let $\mathcal{E}$ be a set of environments that can be combined with S using a combination operator $\otimes$. Let Γ be a set of goals. An *adaptation domain* $\mathcal{A}$ is a set $\mathcal{A} \subseteq \mathcal{E} \times \Gamma$. S can adapt to $\mathcal{A}$, written $S \Vdash \mathcal{A}$ iff for all $(E, \gamma) \in \mathcal{A}$ it holds that $S \otimes E \models \gamma$.

Definition 5 *(Adaptation Space* [28]*)* Let $\mathcal{E}$ be a set of environments that can be combined with S using a combination operator $\otimes$. Let Γ be set of goals. An *adaptation space* $\mathfrak{A}$ is a set $\mathfrak{A} \subseteq \mathfrak{P}(\mathcal{E}, \Gamma)$.

Taken from original publication: Thomas Gabor, Andreas Sedlmeier, Thomy Phan, Fabian Ritz, Marie Kiermeier, Lenz Belzner, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, Marc Zeller, and Claudia Linnhoff-Popien. The scenario coevolution paradigm: adaptive quality assurance for adaptive systems. *International Journal on Software Tools for Technology Transfer*, pages 1–20, 2020

Note that we thus define an adaptation space to be any set of adaptation domains. We can now use the notion of an adaptation space to define a preorder on the adaptivity of any two systems.

Definition 6 *(Adaptation* [28]*)* Given two systems S and S', S' is at least as adaptive as S, written $S \sqsubseteq S'$ iff for all adaptation spaces $\mathcal{A} \in \mathfrak{A}$ it holds that $S \Vdash \mathcal{A} \Longrightarrow S' \Vdash \mathcal{A}$.

Both Definitions 4 and 5 can be augmented to include soft constraints or optimization goals. This means that in addition to checking against Boolean goal satisfaction, we can also assign each system S interacting with an environment E a *fitness* $\phi(S \otimes E) \in F$, where F is the type of fitness values. We assume that there exists a preorder $\preceq$ on F, which we can use to compare two fitness values. We can then generalize Definitions 4 and 5 to respect these optimization goals.

Definition 7 *(Adaptation Domain for Optimization)* Let S be a system. Let $\mathcal{E}$ be a set of environments that can be combined with S using a combination operator $\otimes$. Let Γ be a set of Boolean goals. Let F be a set of fitness values and $\preceq$ be a preorder on F. Let Φ be a set of fitness functions with codomain F. An *adaptation domain* $\mathcal{A}$ is a set $\mathcal{A} \subseteq \mathcal{E} \times \Gamma \times \Phi$. S can adapt to $\mathcal{A}$, written $S \Vdash \mathcal{A}$ iff for all $(E, \gamma, \phi) \in \mathcal{A}$ it holds that $S \otimes E \models \gamma$.

Note that in Definition 7, we only augmented the data structure for adaptation domains but did not actually alter the condition to check for the fulfillment of an adaptation domain. This means that for an adaptation domain $\mathcal{A}$, a system needs to fulfill all goals in $\mathcal{A}$ but is not actually tested on the fitness defined by ϕ. We could define a fitness threshold f we require a system S to surpass in order to adapt to $\mathcal{A}$ in the formalism. But such a check, written $f \preceq \phi(S \otimes E)$, could already be included in the Boolean goals if we use a logic that is expressive enough.

Instead, we want to use the fitness function as soft constraints. We expect the system to perform as well as possible on this metric, but we do not (always) require a minimum level of performance. However, we can use fitness to define a fitness preorder on systems.

Definition 8 *(Optimization)* Given two systems S and S' as well as an adaptation space $\mathcal{A}$, S' is at least as optimal as S, written $S \preceq_{\mathcal{A}} S'$, iff for all $(E, \gamma, \phi) \in \mathcal{A}$ it holds that $\phi(S \otimes E) \preceq \phi(S' \otimes E)$.

Definition 9 *(Adaptation with Optimization)* Given two systems S and S', S' is at least as adaptive as S with respect to optimization, written $S \sqsubseteq^* S'$ iff for all adaptation domains $\mathcal{A} \in \mathfrak{A}$ it holds that $S \Vdash \mathcal{A} \Longrightarrow S' \Vdash \mathcal{A}$ and $S \preceq_{\mathcal{A}} S'$.

In Fig. 1, we introduce a visual representation of systems and the relation of adaptivity given in Definition 9. Note that so far our notions of adaptivity and optimization are purely extensional, which originates from the black-box perspective on adaptation assumed in [28].

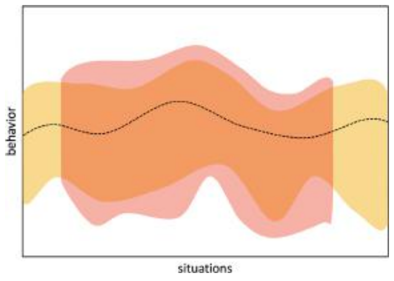


Fig. 1 Illustration of adaptivity according to Definition 9. When the x-axis spans over all possible situations and the y-axis over all possible behaviors, a system like $S_1 = X_1 \rightsquigarrow Y_1$ (orange) or $S_2 = X_2 \rightsquigarrow Y_2$ (red) can be drawn as an area of all the behaviors of S_1 or S_2 so that $S_1 \models \gamma$ or $S_2 \models \gamma$, respectively. For each situation, we show the ideal behavior subject to the fitness ϕ via the dashed black line. S_1 is at least as adaptive as S_2 because it covers at least as many situations as S_1 and performs as least as close to the optimal fitness as S_2 (colour figure online)

3.2 Constructing adaptive systems

We now shift the focus of our analysis a bit away from the question "When is a system adaptive?" toward the question "How is a system adaptive?". This refers to both questions of software architecture (i.e., which components should we use to build an adaptive system?) and questions of software engineering (i.e., which development processes should we use to develop an adaptive system?). We will see that with the increasing usage of methods of machine learning, design-time engineering and run-time adaptation increasingly overlap [46].

Definition 10 *(Adaptation Sequence)* A series of $|I|$ systems $\mathcal{S} = (S_i)_{i \in I}$ with index set I with a preorder $\leq$ on the elements of I is called an *adaptation sequence* iff for all $i, j \in I$ it holds that $i \leq j \Longrightarrow S_i \sqsubseteq^* S_j$

Note that we used adaptation with optimization in Definition 10 so that a sequence of systems $(S_i)_{i \in I}$ that each fulfill the same hard constraints (γ within a singleton adaptation space $\mathfrak{A} = \{\{(E, \gamma, \phi)\}\}$) can form an adaptation sequence iff for all $i, j \in I$ it holds that $i \leq j \Longrightarrow \phi(S_i \otimes E) \preceq \phi(S_j \otimes E)$. This is the purest formulation of an optimization process within our formal framework. Strictly speaking, an optimization *process* would further assume there exists an optimization relation o from systems to systems so that for

Taken from original publication: Thomas Gabor, Andreas Sedlmeier, Thomy Phan, Fabian Ritz, Marie Kiermeier, Lenz Belzner, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, Marc Zeller, and Claudia Linnhoff-Popien. The scenario coevolution paradigm: adaptive quality assurance for adaptive systems. *International Journal on Software Tools for Technology Transfer*, pages 1–20, 2020

all $i, j \in I$ it holds that $i \leq j \implies o(S_i, S_j)$. But for simplicity, we consider the sequence of outputs of the optimization process a sufficient representation of the whole process.

Such an adaptation sequence can be generated by continuously improving a starting system S_0 and adding each improvement to the sequence. Such a task can both be performed by a team of human developers or standard optimization algorithms as they are used in artificial intelligence. Only in the latter case, we want to consider that improvement happening within our system boundaries. Unlike the previously performed black-box analysis of systems, the presence of an optimization algorithm within the system itself does have implications for the system's internal structure. We will thus switch to a more "gray box" analysis in the spirit of [11].

Definition 11 *(Self-Adaptation)* A system S_0 is called *self-adaptive* iff the sequence $(S_i)_{i\in\mathbb{N}, i<n}$ for some $n \in \mathbb{N}$ with $S_i = S_0 \otimes S_{i-1}$ for $0 < i < n$ and some combination operator $\otimes$ is an adaptation sequence.

Please note that we use the term "adaptation" here to mean the improvement in adaptivity as defined in [28]. This is different from some notions of adaptation which allow for a reduction in adaptivity during adaptation as well [1,10]. In our case of adaptation, we can imagine that the system is always able to go back to previous configuration, thus every adaptation only adds to its overall capabilities. To some extent, this already anticipates the perspective of eternal systems which is discussed later in Sect. 9.3 [33].

Note that we could define the property of self-adaptation more generally by again constructing an index set on the sequence (S_i) instead of using $\mathbb{N}$, but chose not to do so to not further clutter the notation. For most practical purposes, adaptation is going to happen in discrete time steps anyway. It is also important to be reminded that despite its notation, the combination operator $\otimes$ does not need to be symmetric and likely will not be in this case, because when constructing $S_0 \otimes S_{i-1}$, we usually want to pass the previous instance S_{i-1} to the general optimization algorithm encoded in S_0. Furthermore, it is important to note that the constant sequence $(S)_{i\in\mathbb{N}}$ is an adaptation sequence according to our previous definition and thus every system is self-adaptive with respect to a combination operator $X \otimes Y =_{\text{def}} X$. However, we can construct non-trivial adaptation sequences using partial orders $\sqsubset$ and $\prec$ instead of $\sqsubseteq$ and $\preceq$. As these can easily be constructed, we do not further discuss their definitions in this paper. In [28], a corresponding definition was already introduced for $\sqsubset$.

The formulation of the adaptation sequence used to prove self-adaptivity naturally implies some kind of temporal structure. So basing said structure around $\mathbb{N}$ implies a very simple, linear and discrete model of time. More complex temporal evolution of systems is also already touched upon in [28]. As noted, there may be several ways to define such a temporal structure on systems. We refer to related and future work for a more intricate discussion on this matter.

So, non-trivial self-adaptation does imply some structure for any self-adaptive system S of type $\mathcal{V} = (V_i)_{i\in I}$: Mainly, there needs to be a subset of the type $\mathcal{V}' \subseteq \mathcal{V}$ that is used to encode the whole relation behind S so that the already improved instances can sufficiently be passed on to the general adaptation mechanism.

For a general adaptation mechanism (which we previously assumed to be part of a system) to be able to improve a system's adaptivity, it needs to be able to access some representation of its goals and its fitness function. This provides a gray-box view of the system. Remember that we assumed a system S could be split into situation X, internal state Z and behavior Y, written $S = X \overset{Z}{\rightsquigarrow} Y$. If S is self-adaptive, it can form a non-trivial adaptation sequence by improving on its goals or its fitness. In the former case, we can now assume (that there exists some relation $G \subseteq X \cup Z$ so that $S \models \gamma \iff G \models \gamma$ for a fixed γ in a singleton-space adaptation sequence. In the latter case, we can assume that there exists some relation $F \subseteq X \cup Z$ so that $\phi(S) = \phi(F)$ for a fixed ϕ in a singleton-space adaptation sequence. Effectively, if we employ a general mechanism for self-adaptation, as it is commonly done in current applications of machine learning, it is necessary that the result of the adaptation is passed back into the system.

Obviously, when we want to construct larger self-adaptive systems using self-adaptive components, the combination operator needs to be able to combine said sub-systems G and/or F as well. In the case where the components' goals and fitnesses match completely, the combination operator can just use the same sub-system twice. However, including the global goals or fitnesses within each local component of a system does not align with common principles in software architecture (such as encapsulation) and does not seem to be practical for large or open systems (where no process may ensure such a unification). Thus, constructing a component-based self-adaptive system requires a combination operator that can handle potentially conflicting goals and fitnesses. We again define such a system for a singleton adaptation space $\mathfrak{A} = \{(E, \gamma, \phi)\}$ and leave the generalization to all adaptation spaces out of the scope of this paper.

Definition 12 *(Multi-Agent System)* Given a system $S = S_1 \otimes \cdots \otimes S_n$ that adapts to $\mathcal{A} = \{(E, \gamma, \phi)\}$. Iff for each $1 \leq i \leq n$ with $i, n \in \mathbb{N}, n > 1$ there is an adaptation domain $\mathcal{A}_i = \{(E_i, \gamma_i, \phi_i)\}$ so that (1) $E_i = E \otimes S_1 \otimes \cdots \otimes S_{i-1} \otimes S_{i+1} \otimes \cdots \otimes S_n$ and (2) $\gamma_i \neq \gamma$ or $\phi_i \neq \phi$ and (3) S_i adapts to $\mathcal{A}_i$, then S is a *multi-agent system* with agents $S_1, \ldots, S_n$.

It is important to note here that the combination operator $\otimes$ may again be arbitrarily complex and does not need to work the same way for the construction of S and the construction of E_i above. The definition of a multi-agent system

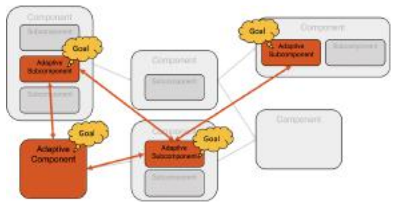


Fig. 2 Illustration of the emergence of an (implicit) multi-agent system in a fictitious architecture of software components. Adaptive components interact by manipulating their environment to achieve their individual goals

only requires the decomposability of the respective systems with respect to some $\otimes$. Obviously, the notion then varies in expressiveness and explanatory power depending on the choice of $\otimes$.

For practical purposes, we usually want to use the notion of multi-agent systems in a transitive way, i.e., we can call a system a multi-agent system as soon as any part of it is a multi-agent system according to Definition 12. Formally, S is a multi-agent system if there are systems components S', R so that $S = S' \otimes R$ and S' is a multi-agent system. We argue that this transitivity is not only justified but a crucial point for systems development of adaptive systems: Agents tend to utilize their environment to fulfill their own goals and can thus "leak" their goals into other system components (see Fig. 2). Note that Condition (2) of Definition 12 ensures that not every system constructed by composition is regarded a multi-agent system; it is necessary to feature agents with (at least slightly) differing adaptation properties.

For the remainder of this paper, we will apply Definition 12 "backwards." Whenever we look at a self-adaptive system S, whose goals or fitnesses can be split into several sub-goals or sub-fitnesses, we can regard S as a multi-agent system. Using this knowledge, we can apply design patterns from multi-agent systems to all self-adaptive systems without loss of generality. Furthermore, we need to be aware that especially if we do not explicitly design multi-agent coordination between different sub-goals, such a coordination will be done implicitly. Essentially, there is no way around generalizing software engineering approaches for self-adaptive systems to potentially adversarial components.

4 Example domain

To illustrate the definitions of the previous section, we introduce an example system called *Grid World Smart Factory*, which has also been used and implemented in [24]. However, we first introduce a formal definition of a system for this domain.

4.1 Setup

An instance of the smart factory domain contains a number of *items* that have to be processed at *workstations* of different types, while avoiding collisions with dynamically placed *obstacles*. The system is tasked with navigating a robotic agent through the smart factory in order to eventually process all the items.

In our example, the smart factory uses a discrete grid of size 7×8, as shown in Fig. 3. Thus, possible positions for entities of any kind within the factory are all $p \in P$, where $P = \{1, \ldots, 7\} \times \{1, \ldots, 8\}$.

Five workstations $W \subset P$ are placed at fixed positions so that $W = \{w_1, \ldots, w_5\} = \{(1, 5), (4, 7), (5, 1), (6, 3), (6, 7)\}$. Each workstation w is assigned a fixed type $t(w)$, $t : W \rightarrow \{red, green, blue\}$, so that $t(w_1) = blue$, $t(w_2) = red$, $t(w_3) = red$, $t(w_4) = green$, $t(w_5) = green$.

The domain is parametric on the position of four obstacles $O = \{o_1, o_2, o_3, o_4\} \subset P$ so that $O \cap W = \emptyset$.

A robotic agent r is given via its current position $r \in P$. Note that the starting position of the robot always is $r = (1, 1)$. The robotic agent is able to execute four movement actions $v \in V = \{v_\vartriangle, v_\triangleright, v_\triangledown, v_\triangleleft\}$, $v : P \rightarrow P$, where for all $\circ \in \{\vartriangle, \triangleright, \triangledown, \triangleleft\}$, we define

$$v_\circ(y, x) = \begin{cases} v'_\circ(y, x) & \text{if } v'_\circ(y, x) \in P \setminus (W \cup O) \\ (y, x) & \text{otherwise} \end{cases} \quad (1)$$

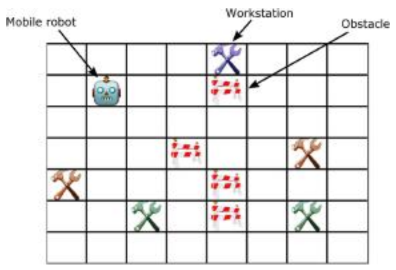


Fig. 3 Visualization of the smart factory domain. A mobile robot can travel north, east, south and west on the grid. It needs to visit workstations in order to retrieve items and then needs to visit other workstations in order to process these items. Attempting to walk out of the grid, into a workstation or into an obstacle is penalized. Obstacle positions vary according to the setting of the scenario x

Taken from original publication: Thomas Gabor, Andreas Sedlmeier, Thomy Phan, Fabian Ritz, Marie Kiermeier, Lenz Belzner, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, Marc Zeller, and Claudia Linnhoff-Popien. The scenario coevolution paradigm: adaptive quality assurance for adaptive systems. *International Journal on Software Tools for Technology Transfer*, pages 1–20, 2020

where all $v'_\circ$ are naturally defined as $v'_\triangle(y,x) = (y-1, x)$, $v'_\triangleright(y,x) = (y, x+1)$, $v'_\triangledown(y,x) = (y+1, x)$, $v'_\triangleleft(y,x) = (y, x-1)$. Note that any action that returns using the *otherwise* branch in Eq. 1 is called *illegal*. Any action that is not illegal is called *legal*. It follows that when the agent position r is only altered via the application of actions, it always holds that $\{r\}$, O, and W are fully disjunct.

Given a position $p \in P$, we define the neighborhood of p as $N : P \to \mathfrak{P}(P)$ with

$$N(y,x) = \{v_\circ(y,x) \ : \ \circ \in \{\triangle, \triangleright, \triangledown, \triangleleft\}\}. \tag{2}$$

Since all actions are reversible, $N(y,x)$ both contains all position that can be reached from (y,x) and all position that (y,x) can be reached from.

We call an instance of the smart factory domain *valid* iff the agent can reach new positions from its initial position and all workstations can be reached, i.e.,

$$\forall e \in \{r\} \cup W : N(e) \neq \emptyset. \tag{3}$$

Note that this simple test suffices since we only have four obstacles and no two workstations are next to each other, so the only way to make any of them inaccessible is to place all four obstacles around it.

We define an item as a tuple m containing a current position and a series of workstation types, i.e., $m = (p, \langle t_i \rangle_{i \in I})$ where $p \in P$ and I is an index set and for all $i \in I$ it holds that $t_i \in \{\mathit{red}, \mathit{green}, \mathit{blue}\}$. Semantically, an item needs to visit workstations of all the given types t_i in the given order in order to be fully processed. As long it is not fully processed, it poses a task to the system. Our system is tasked to produce five items $M = \{m_1, \dots, m_5\}$. When all of these are fully produced, the domain instance is finished successfully. Note that initially, items are placed at workstations. We thus define the current position of an item m as $c(m)$ with $c : M \to P$. Furthermore, up to one item can be carried by the mobile agent so that $c(m) = r$. Note that it always holds for all $m \in M$ that $c(m) \in \{r\} \cup W$, i.e., no items can be left on the factory floor. We also define the function $b : W \to \langle M \rangle$, which is given a workstation and returns a sequence of items so that for all workstations $w \in W$ and all items $m \in b(w)$ it holds that $c(m) = w$. The first item of that sequence is the one that can be picked up next at the respective workstation.

In our setup, we use the items

$$m_1 = ((1,5), \langle \mathit{green}, \mathit{red} \rangle),$$
$$m_2 = ((1,5), \langle \mathit{green} \rangle),$$
$$m_3 = ((4,7), \langle \mathit{red}, \mathit{blue}, \mathit{green} \rangle),$$
$$m_4 = ((1,5), \langle \mathit{green}, \mathit{blue} \rangle),$$
$$m_5 = ((6,3), \langle \mathit{red} \rangle).$$

Of course, we now need to augment our previously defined set of movement actions $V = \{v_\triangle, v_\triangleright, v_\triangledown, v_\triangleleft\}$ to allow for interaction with items. We thus define the set of actions $A = \{a_\triangle, a_\triangleright, a_\triangledown, a_\triangleleft, a_\boxplus, a_\boxminus\}$ so that for all $\circ \in \{\triangle, \triangleright, \triangledown, \triangleleft, \boxplus, \boxminus\}$ and $a_\circ : P \times \mathfrak{P}(P \times \langle T \rangle) \to P \times \mathfrak{P}(P \times \langle T \rangle)$ it holds that

$$a_\circ(r, M) = \begin{cases} (v'_\circ(r), M) & \mathit{if} \circ \in \{\triangle, \triangleright, \triangledown, \triangleleft, \} \\ & \mathit{and}\ a'_\circ(r) \in P \setminus (W \cup O) \\ (r, \{m'_\boxplus\} \cup M \setminus m) & \mathit{if} \circ = \boxplus \\ & \mathit{and}\ c(m) \in N(r) \\ & \mathit{and}\ \forall \hat{m} \in M : c(\hat{m}) \neq r \\ (r, \{m'_\boxminus\} \cup M \setminus m) & \mathit{if} \circ = \boxminus \\ & \mathit{and}\ c(m) = r \\ & \mathit{and}\ t(w) = t_1 \\ (r, M) & \mathit{otherwise} \end{cases} \tag{4}$$

where $m = (p, \langle t_1, \dots, t_n \rangle)$ is any element from M, w is any element from $W \cap N(r)$ and subsequently $m'_\boxplus = (r, \langle t_1, \dots, t_n \rangle)$ and $m'_\boxminus = (w, \langle t_2, \dots, t_n \rangle)$. We implicitly quantify existentially over all $m \in M$. The function $a_\circ$ still remains deterministic only because the conditions are formulated so that at most one $m \in M$ fits them in our setup. In the more general setup, it would be valid to pick any arbitrary option. For $w \in W \cap N(r)$, again, we implicitly quantify, although it only matters in the third case. Again, this quantification can yield at most one element as no two workstations of the same type have shared neighboring positions in our setup. For the more general case, we can simply pick a w at random should multiple assignments validate this condition here. Finally, note that when an item is fully processed, we assume $m'_\boxminus = (w, \langle\rangle)$ for some position $w \in P$, i.e., we keep all the processed items "lying around" with an empty task list. We could also choose to remove fully processed items entirely from the system by specifying $a_\circ(r, M) = (r, M \setminus m)$ in that case. Since we used the power set $\mathfrak{P}(T \times \langle T \rangle)$ or the type of $a_\circ$, we are flexible in that choice. For ease of definition, we will later fix the amount of items present in the adaptive system, favoring the "lying around" approach.

Again, every action that results from taking the *otherwise* branch of $a_\circ$ is called illegal. The action $a_\boxplus$ is called *pick-up* and the action $a_\boxminus$ is called *drop-off*.

4.2 Adaptive system

Having defined the complete setup of our smart factory domain, we can now proceed to define the adaptation domain.

Taken from original publication: Thomas Gabor, Andreas Sedlmeier, Thomy Phan, Fabian Ritz, Marie Kiermeier, Lenz Belzner, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, Marc Zeller, and Claudia Linnhoff-Popien. The scenario coevolution paradigm: adaptive quality assurance for adaptive systems. *International Journal on Software Tools for Technology Transfer*, pages 1–20, 2020

We define the system $S = X \overset{Z}{\rightsquigarrow} Y$ where X is a list of elements $\langle x_t \rangle_{0<t<n}$ with the maximum execution length $n \in \mathbb{N}$ (and likewise for Y). Note that without loss of generality, we can assume that all execution traces are of the same length n by simply setting n to the maximum length and filling up shorter paths with "nil" elements. We set

$$x = \langle r_t, M_t, O_t \rangle \tag{5}$$

with robot position $r_t \in P$, item list $M_t \in (P \times \langle T \rangle)^5$ and obstacles $O_t \in P^4$. Note that we specify a fixed amount of 5 items that may thus be present in the system. We also specify

$$y_t = \langle a_t \rangle \tag{6}$$

with action $a_t \in \{a_\triangle, a_\triangleright, a_\triangledown, a_\triangleleft, a_\boxplus, a_\boxminus\}$. The legal elements for X and Y are defined by the type of the system's policy $\Pi : (P \times (P \times \langle T \rangle)^5 \times P^4)^n \rightarrow \{a_\triangle, a_\triangleright, a_\triangledown, a_\triangleleft, a_\boxplus, a_\boxminus\}^n$ where $n \in \mathbb{N}$ is the maximum execution time of the system so that

$$Y = \langle \Pi(X) \rangle. \tag{7}$$

We omit any further specification on the policy π at hand (and accordingly for the internal state Z) as the policy is the core of the system's implementation, which we discuss in more detail in Sects. 5 and 8 .

However, we can use the given definition of the system's interface to specify its adaptation domain. We define a static environment E, which means that once a system $S \otimes E$ is composed, the environment does not change or react to the system's actions. In our example, the environment consists of the obstacles' position, so

$$E = \{o_1, o_2, o_3, o_4\} \tag{8}$$

with $o_i \in P$ for all $i = 1, .., 4$. Note that we could also write $E = \emptyset \rightsquigarrow \langle \{o_1, o_2, o_3, o_4\} \rangle_{0<t<n}$ to adhere to the previously introduced notation. We then define the composed system $S \otimes E$ to use the obstacles given by E to set all respective inputs X so that for all x_t in $X = \langle x_t \rangle_{0<t<n}$ it holds that $x_t = \langle r_t, M_t, \{o_1, o_2, o_3, o_4\} \rangle$ for some r_t, M_t.

At this stage, we might just as well-define a dynamic environment that could change the obstacles' positions over time by setting $E = \emptyset \rightsquigarrow \langle \{o_{1,t}, o_{2,t}, o_{3,t}, o_{4,t}\} \rangle_{1<t<n}$ with $o_{i,t}$ depending on the current step of system execution t. A reactive environment $E = \langle r \rangle_{1<t<n} \rightsquigarrow \langle \{o_{1,t}, o_{2,t}, o_{3,t}, o_{4,t}\} \rangle_{1<t<n}$ might even change any obstacle's position $o_{i,t}$, for example with respect to the robot position r according to some environment policy $\rho : P \times \mathbb{N} \rightarrow P$ so that $o_{i,t} = \rho(r_t, i)$. However, we will omit further considerations on dynamic environments for brevity and will resort to a static environment for the running example in this paper.

However, please note that we can still generate many different static environments to be part of the adaptation domain. This will require the system to be able to handle various configurations of non-moving obstacles but not require the system to be able to handle moving obstacles.

We can now define a simple system goal such as

$$\gamma(S) \iff \exists t \ : \ \forall m \in M_t \ : \mathit{finished}(m) \tag{9}$$

where M_t is given via $x_t = \langle r_t, M_t, O_t \rangle$ (coming from $S = X \rightsquigarrow Y$ and $X = \langle x_t \rangle_{0<t<n}$) as in Eq. 5 and $\mathit{finished} : P \times \langle T \rangle \rightarrow \mathbb{B}$ is given via

$$\mathit{finished}(\mathit{pos}, \mathit{tasks}) \iff \mathit{tasks} = \langle \rangle. \tag{10}$$

Semantically, $\gamma(S)$ holds iff at some point during the execution, all items in the system have been processed. Note that we use a very raw formulation for a property that might be more fittingly expressed in some temporal logic. But using simple predicate logic is sufficient for the present running example. A different goal function might be to never execute an illegal action, which might be written as $\gamma'(S) \iff \forall t \ : \ \neg \mathit{illegal}(a_t)$. For the running example, we will focus on the single goal function γ, though.

The definitions made in this subsection now allow us to finally define an adaptation domain such as $\mathcal{A} = \{(((2, 5), (4, 4), (5, 5), (6, 5)), \gamma)\}$, which defines the environment setup shown in Fig. 3 and the goal function of Eq. 10. For our running example, we want the system to work for any arbitrary (legal) configuration of obstacle position so that we define

$$\mathcal{A} = \{((o_1, o_2, o_3, o_4), \gamma) \ : \ i = 1, \ldots, 4, \\ o_i \in P \setminus W \setminus \{(1, 1)\}\}. \tag{11}$$

We can now further augment this declaration to include an optimization target (as given in Definitions 7 and 8). Using $\mathbb{N}$ as the space of the fitness values and $\geq$ as a preorder (meaning that me minimize the fitness value) we can define a fitness function

$$\phi(S \otimes E) = \min \{t \in \mathbb{N} \mid \ \forall m \in M_t \ : \ \mathit{finished}(m)\} \tag{12}$$

where M_t is given via $x_t = \langle r_t, M_t, O_t \rangle$ (coming from $S = X \rightsquigarrow Y$ and $X = \langle x_t \rangle_{0<t<n}$) as in Eq. 5 and $\mathit{finished} : P \times \langle T \rangle \rightarrow \mathbb{B}$ is given via Eq. 10. The fitness function ϕ as defined in Eq. 12 then returns the amount of time steps the system took to reach the *finished* predicate, i.e., the time it took to fully process all items. This would be a typical target for minimization. Note that in this case, there exists a clear correspondence between the goal function γ and the fitness

Taken from original publication: Thomas Gabor, Andreas Sedlmeier, Thomy Phan, Fabian Ritz, Marie Kiermeier, Lenz Belzner, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, Marc Zeller, and Claudia Linnhoff-Popien. The scenario coevolution paradigm: adaptive quality assurance for adaptive systems. *International Journal on Software Tools for Technology Transfer*, pages 1–20, 2020

function ϕ as only systems that fulfill γ have a finite value for ϕ.

Different reasonable fitness functions exist: For example, we may want to get rid of the goal function entirely and instead formulate a fitness function that maximizes the amount of that are fully processed (instead if enforcing that all of them are eventually fully processed always). Or we may want to optimize an entirely different goal like minimizing the turns of direction the agent is taking.

In the end, setting the right γ and ϕ for the adaptation domain is a decision to be made in system design and is crucial to fulfilling the initial requirements. In particular, the interaction between the goal and the fitness function is to be considered.

Having given an adaptation domain, we can write $S \Vdash \mathcal{A}$ iff the system S can adapt to $\mathcal{A}$, i.e., S fulfills the goal function for all respective environments in $\mathcal{A}$. We can also trivially define a *singleton adaptation space*

$$\mathfrak{A} = \{\mathcal{A}\}, \tag{13}$$

which shall suffice for the example given here.

5 Implementation of adaptation

So far we constructed a framework to compare the degree of adaptivity of two given systems. In this section, we discuss how to give these adaptive systems. This boils down to the problem: Given a system S, how can its adaptivity be improved, i.e., how can we generate a system S' so that $S \sqsubseteq S'$. The art of generating (software) systems is called (software) engineering. Traditionally, we would specify higher adaptivity as a requirement and task a group of software developers to improve the system S. They would then write code to cover additional adaptation domains (within the given adaptation space) or improve the system's performance on a given fitness function (when considering optimization) as follows from Definition 9.

5.1 Adaptation via machine learning

Newer methods in software engineering aim to automate (parts of) that process [7,12,46]. The most trivial means of automation is probably stochastic search. For this, we require a variation operator $vary : \mathcal{R}(\mathcal{V}) \to \mathcal{R}(\mathcal{V})$ where $\mathcal{R}(\mathcal{V})$ is the domain of all relations over $\mathcal{V}$ and $\mathcal{V}$ is a type of system and $S \in \mathcal{R}(\mathcal{V})$ (see Definition 1). Note that *vary* is not a function but $vary(S)$ returns a random variant of a given system S any time it is executed. Usually, $vary(S)$ will not generate new systems from scratch but reuse almost all parts of S and just implement small changes to it. We can then run a stochastic search process as shown in Algorithm 1. Note that aside from the *vary* operator, we also need to provide a *termination_criterion* that allows us to stop the search once a sufficient solution has been found or we have spent too much time on searching. The operator $chance : [0; 1] \subset \mathbb{R} \to \mathbb{B}$ can be defined generally to return *true* only with the given chance and *false* otherwise. Further note that computing $S \sqsubseteq S'$ can become very expensive or even infeasible for sufficiently complex systems S, S'. We later show in Sects. 7 and 8 how to construct a set of more concrete test cases against which such properties can be evaluated more efficiently, but only while losing out on the exactness of the result. In general, sampling is usually employed to approximate such properties on large domains.

Algorithm 1 Stochastic Search

```
Require: system $S$
1: while $\neg termination\_criterion$ do
2:     $S' \leftarrow vary(S)$
3:     if $S \sqsubseteq S' \vee chance(\varepsilon)$ then
4:         $S \leftarrow S'$
5:     end if
6: end while
7: return $S$
```

Algorithm 2 Stochastic Search in Parameter Space

```
Require: system $S$, initial parameter $\theta_0$
1: $\theta \leftarrow \theta_0$
2: while $\neg termination\_criterion$ do
3:     $\theta' \leftarrow vary(\theta)$
4:     if $S \otimes \theta' \sqsubseteq S \otimes \theta \vee chance(\varepsilon)$ then
5:         $\theta \leftarrow \theta'$
6:     end if
7: end while
8: return $S \otimes \theta$
```

What makes stochastic search of this form generally infeasible is that more adaptive systems are typically very rare among all system variants that can be generated via *vary*. We thus need to restrict the possible variations to somewhat meaningful systems at least. Most commonly, we do this by fixing most components of the system S and introducing a parameterization θ of some type Θ describing important aspects of the system's behavior. Stochastic search then only needs to search the much more abstract parameter space Θ. When given a variation operation $vary : \Theta \to \Theta$ and a (usually random) initial value $\theta_0 \in \Theta$, we can rewrite Algorithm 1 to search for the correct parametrization as seen in Algorithm 2. In a machine learning setting, the system S could typically include a neural network whose weights are encoded in θ. This way, the weights space is relatively small compared to altering the whole system but as long as the neural network's outputs are important to the system behavior, it can be heavily influenced by just changing the weights.

Obviously, we can still spend a lot of time sampling randomly varied settings for θ without ending up with any good solutions. We can usually shorten the search process if we

Taken from original publication: Thomas Gabor, Andreas Sedlmeier, Thomy Phan, Fabian Ritz, Marie Kiermeier, Lenz Belzner, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, Marc Zeller, and Claudia Linnhoff-Popien. The scenario coevolution paradigm: adaptive quality assurance for adaptive systems. *International Journal on Software Tools for Technology Transfer*, pages 1–20, 2020

Algorithm 3 Gradient Descent in Parameter Space

```
Require: system $S$, initial parameter $\theta_0$, update rate $\alpha$
1: $\theta \leftarrow \theta_0$
2: while $\neg$termination_criterion do
3:     $\theta \leftarrow \alpha \cdot \nabla\theta$
4: end while
5: return $S \otimes \theta$
```

Algorithm 4 Gradient Descent with Sampling in Parameter Space

```
Require: system $S$, initial parameter $\theta_0$, update rate $\alpha$,
    training data set $D$
1: $\theta \leftarrow \theta_0$
2: while $\neg$termination_criterion do
3:     $x, y \leftarrow sample(D)$
4:     $\theta \leftarrow \alpha \cdot \nabla\theta(x, y)$
5: end while
6: return $S \otimes \theta$
```

can compute a gradient for a specific point θ in the parameter space. Note that this is generally not the case in our setting: We want to improve the system's adaptivity by following the "at least as adaptive as" relation $\sqsubseteq$, which is defined on subset inclusion and thus naturally discrete. Intuitively, we can recognize if system S' is at least as adaptive as S, but we have no notion of how much more adaptive it is. However, we can resort to the case of adaptation with optimization (see Definition 9): On some fitness value types F, we can define a gradient. In the case of neural networks, e.g., $F = \mathbb{R}^n$ for some $n \in \mathbb{N}$ and for a given fitness value $f = \phi(S \otimes \theta)$ with fitness function ϕ, we can compute the gradient $\nabla\theta = \nabla\phi(S \otimes \theta)$.

In order to find a good setting for the parameter θ, we can then use a more direct approach to search like gradient descent. As shown in Algorithm 3, when we can compute the gradient, we can use it to update the parameter θ to the most promising direction. The update rate $\alpha \in (0; 1) \subset \mathbb{R}$ controls how far along the gradient we go with each iteration.

Backpropagation is a variant of gradient descent specifically fitted to update the weights of neural networks. For more details on the method, we refer to other work [20,35,37].

Of course, computing $\phi(S \otimes \theta)$ tends to be non-trivial. If we have a precise model of what makes the system perform well according to ϕ, we can usually just build this behavior into the system and do not require elaborate and expensive search algorithms. It is important to note that, in the general case, no search algorithm can effectively beat random search. This is called the *No Free Lunch Theorem* [47]. However, we can always build into the search as much knowledge about the structure of the problem as we have, which then allows us to get better results for problems matching that knowledge. In the typical use case for machine learning, we do not have complete knowledge about how a good system should look like but we have single evaluation points far and between, telling us about concrete instantiations for θ and the respective value of $\phi(S\otimes\theta)$. Machine learning is the task of building a model from these data points.

For example, let us consider a visual system that needs to recognize if a given picture x contains a cat or not. This system might use a neural network with weights θ and we are looking for a $\theta \in \Theta$ that makes the system recognize images of cats. For that search, we need a set of training data $D = \{\langle x_1, y_1\rangle, \dots, \langle x_n, y_n\rangle\}$ where for all $i \in [1; n] \subset \mathbb{N}$ it holds that x_i is a image from the set of all images X and $y_i = 1$ iff x_i contains a cat, $y_i = 0$ otherwise. We can then compute the fitness

$$\phi(S \otimes \theta) = \sum_{i=1}^{n} |Y(x_i) - y_i| \tag{14}$$

where $Y(x_i)$ is given via $S \otimes \theta = x_i \rightsquigarrow Y(x_i)$. When the set of training data is large and diverse enough, we assume that the parameter θ that works best on the training data, also works best (or at least well) on new, unseen data.

Note that typically, we do not evaluate each solution candidate for θ on the whole training set but for performance reasons opt for a more gradual process as shown in Algorithm 4, where $\nabla\theta(x, y) = \phi(S \otimes \theta, x, y)$ and $\phi : \mathcal{V} \times X \times Y \rightarrow F$ is given via

$$\phi(S \otimes \theta, x, y) = |Y(x) - y| \tag{15}$$

where $Y(x)$ is defined as for Eq. 14. When doing so, we usually need more iterations of the whole process (i.e., a more lenient *termination_criterion*) but each evaluation of ϕ is much less computationally expensive. This approach represents the common ground for techniques like supervised machine learning or reinforcement learning [20,41].

Methods as shown in Algorithms 1–4 have implications for software engineering: When applying machine learning, we are not certain of the exact system that we will end up with, which, in fact, is the whole purpose of machine learning: to not exactly figure out the full system. This buys some immense possibilities to create complex behavior and adapt to a wide range of situations. However, it also introduces new tasks into the workflow of programming systems.

5.2 Software engineering for machine learning

Figure 4 shows an engineering process for machine learning. At the top blue level, we see typical phases used in process models for classical software engineering. They provide an orientation about what activities new machine learning tasks can be compared to. Note that we assume an agile development process anyway: The whole process shown in Fig. 4 is not necessarily run in sync with the

Taken from original publication: Thomas Gabor, Andreas Sedlmeier, Thomy Phan, Fabian Ritz, Marie Kiermeier, Lenz Belzner, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, Marc Zeller, and Claudia Linnhoff-Popien. The scenario coevolution paradigm: adaptive quality assurance for adaptive systems. *International Journal on Software Tools for Technology Transfer*, pages 1–20, 2020

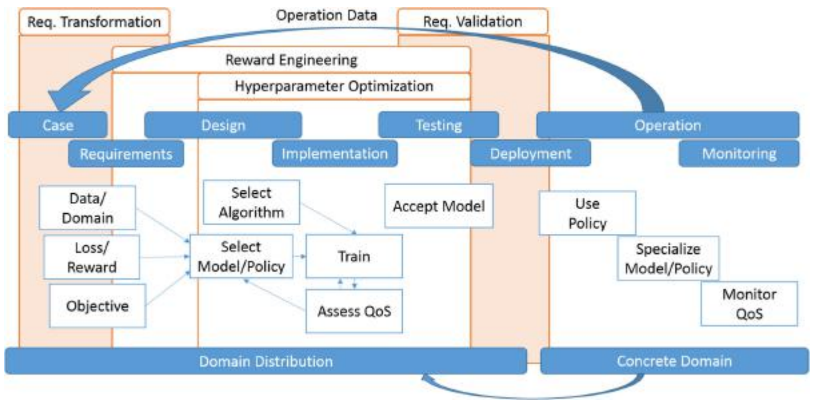


Fig. 4 Machine learning pipeline. Split between classical phases of system development, we can see the central activities necessary for the successful application of machine learning

development process of the rest of the system (which we still assume to be programmed in a mostly classical way). Instead, the process of engineering machine learning could be run several times (as a sprint, e.g.,) within a single activity in a surrounding development process. This is why we will put observations made during the operation of the resulting system (called "operation data" here) into the case and requirement phases of the next iteration of the machine learning pipeline (as symbolized by the large blue arrow).

At the bottom blue level, we discern show the domain within which the individual tasks take place. The first parts of the machine learning pipeline operate on a domain distribution, i.e., they are not specialized on a single instance of a use case but are designed to find models and solutions general enough to work on a range of similar tasks. Even when we only target a single domain eventually, having a decent amount of diversity during training is crucial to the success of machine learning [6,24,43]. During deployment, we switch from the more general distribution of possible domains to a more concrete instantiation fed with all the information we have about the deployed system and the environment it is deployed in. Again, whenever we observe our original assumptions on the distribution of domains to be flawed, we feed back gained knowledge into the next iteration of the machine learning pipeline.

This handling of domains closely mirrors the definition of the adaptation space $\mathfrak{A}$: Recall that in order to build a more adaptive system S, it needs to be able to adapt to larger subset of the adaptation space (or adapt to the same subset better) as stated in Definition 9. Thus, when designing the autonomous adaptation mechanisms in the first part of the machine learning pipeline, we in fact operate on the whole adaptation space $\mathfrak{A}$. However, when it comes to building a concrete system, we will only face a single adaptation domain $\mathcal{A} \in \mathfrak{A}$ at once, perhaps in succession.

We will now briefly discuss each task appearing in the machine learning pipeline (again cf. Fig. 4). They are depicted by the white boxes with a blue border. Some of them are grouped into logical phases using orange boxes.

Data/domain In order to even begin a case description, we need to assure that we have a sufficiently detailed description of the domain we want to use the system in (as given by the definition of environments $\mathcal{E}$ within the adaptation space $\mathfrak{A}$ as shown in Sect. 4.1). Also note that many machine learning algorithms require large amount of high-quality data, which then needs to be provided alongside or instead a full domain description.

Loss/reward This artifact is also included in the adaptation space. The definition and usage of the fitness function ϕ maps exactly to the use of loss or reward functions in most machine learning approaches. It needs to be defined accurately at the beginning of the machine learning pipeline.

Objective This artifact maps to the goals γ within the adaptation space $\mathfrak{A}$. As discussed, in many cases, the fitness function will be derived from the goals or at least altered to support their fulfillment. However, there also often are additional goals which cannot be expressed in the fitness function alone, for example, because they are hard constraints on system

Taken from original publication: Thomas Gabor, Andreas Sedlmeier, Thomy Phan, Fabian Ritz, Marie Kiermeier, Lenz Belzner, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, Marc Zeller, and Claudia Linnhoff-Popien. The scenario coevolution paradigm: adaptive quality assurance for adaptive systems. *International Journal on Software Tools for Technology Transfer*, pages 1–20, 2020

safety that cannot be opened up to optimization. In this case, the goals γ need to be derived from the fitness function.

Select model/policy In this task, we need to define what parts of the system should actually be adapted using machine learning techniques. In case of supervised learning, we are usually speaking of a model representing the data; in the case of reinforcement learning, we use the word policy to refer to a way to encode behavior. Either way, the definition of the model (for example, using a policy network returning the next action of the system) is the biggest influence on the choice of the parameter space Θ (cf. Sect. 5.1).

Select algorithm Knowing which parameter space Θ is to be optimized often aids in the choice of a (possibly highly specialized) optimization algorithm. A choice of (concrete instances of) Algorithms 1–4 might be made here.

Train During the training task, the algorithm selected is applied to optimize the parameters $\theta \in \Theta$ for the selected model or policy. In (hopefully) all cases, this task will be performed automatically by a computer. However, it is usually very resource-intensive and thus requires a lot of manual tweaking: Setting up the right hardware/software platforms, choosing the right meta-parameters (maximum run-time, minimum success, parallelization, etc.) and so on.

Assess QoS Usually, reward yield or loss reduction are used as metrics during training automatically. However, most machine learning algorithms are highly stochastic in nature. Thus, we suggest a separate task for the assessment of the quality of service provided by the automatically trained system. At this stage, we may filter out (and/or redo) bad runs or and check if our problem formulation and selection of algorithms and data structures were sufficient to get the desired quality of a solution.

Accept model As shown in Fig. 4, the tasks involved in the selection of models/policies, training and assessing the quality of the returned solutions form a typical feedback loop. Part of the accept model task is to decide when to break this loop and what model/policy (usually represented by the parameters θ) to return. Usually, we will return the best policy according to the quality of service assessment, but there may be cases where we want to return multiple policies (like a Pareto front, e.g.,).

Use policy Once a suitable model/policy has been found, we assume that deployment happens the same way as for classical systems. At this task, we are thus ready to execute the behavior of the system as given by the model/policy. Note that formally, executing the system S with model/policy θ in a concrete domain $\mathcal{A}$ corresponds to computing $S \otimes \theta \Vdash \mathcal{A}$.

Specialize model/policy As previously discussed, the training loop has not been executed on the deployed domain $\mathcal{A}$ but on a distribution of domains drawn from the adaptation space $\mathfrak{A}$. When we recognize that $\mathcal{A}$ is not going to be subject to substantial changes any more, it makes sense to specialize on the concrete domain instance. This can be done through classical means (adding specialized behavior, removing now inaccessible program parts) or through means of machine learning (re-running a training feedback loop but based on the experiences generated in $\mathcal{A}$ instead of $\mathfrak{A}$). In the latter case, we could actually enter a complete other instantiation of the machine learning pipeline.

Monitor QoS Even when training and assessment have shown that our system $S \otimes \theta$ does fulfill our quality goals, it is most important to continually monitor that property throughout operations. Mistakes in the definition of (the parts of) $\mathfrak{A}$ or general changes in the domain, including subtle phenomena like drift, may cause the trained system to be incapable of further operation. In order to prevent this and re-train as early as possible, we need not only to monitor the defined metrics of quality of service directly, but also keep an eye out for indicators of upcoming changes in quality, for example through means of anomaly detection [30].

It is clear that the machine learning pipeline discussed in this section has no claim of completeness. Many tasks could be changes or added to it. We introduced the pipeline to show that while some necessary changes to the software engineering process closely mirror tasks for classical systems, others introduce entirely new challenges and shift the focus where the main work of software developers should fall. We will use this analysis as a foundation to sum up the major changes into core concepts in the following section.

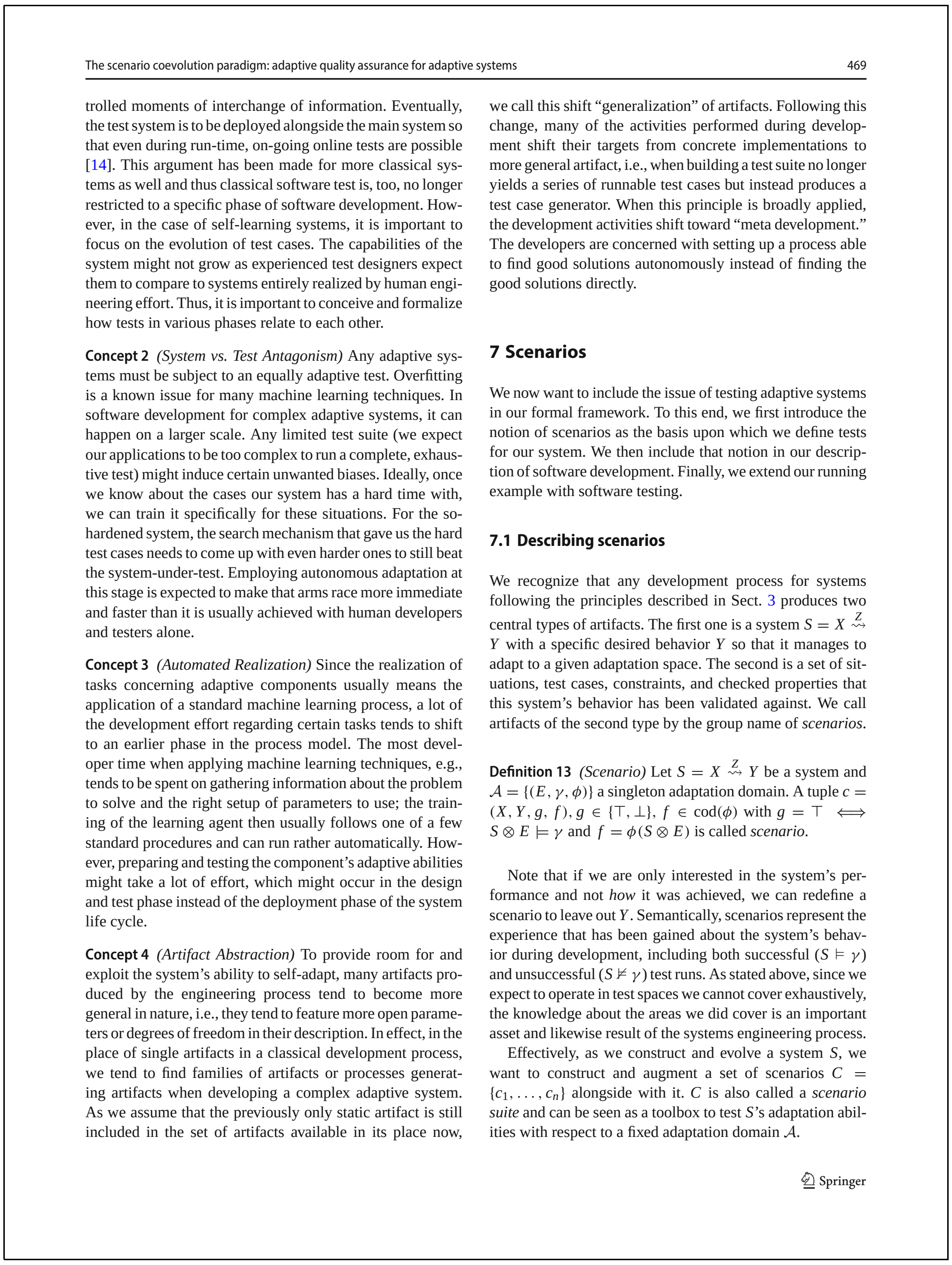
6 Core concepts of adaptive software engineering

Literature makes it clear that one of the main issues of the development of self-adapting systems lies with *trustworthiness*. Established models for checking systems (i.e., verification and validation) do not really fit the notion of a constantly changing system. However, these established models represent all the reason we have at the moment to trust the systems we developed. Allowing the system more degrees of freedom thus hinders the developers' ability to estimate the degree of maturity of the system they design, which poses a severe difficulty for the engineering progress, when the desired premises or the expected effects of classical engineering tasks on the system-under-development are hard to formulate.

To aid us control the development/adaptation progress of the system, we define a set of *core concepts*, which are basically patterns for process models. They describe the paradigm shifts to be made in the engineering process for complex, adaptive systems in relation to more classical models for software and systems engineering.

Concept 1 *(System and Test Parallelism)* The system and its test suite should develop in parallel from the start with con-

Taken from original publication: Thomas Gabor, Andreas Sedlmeier, Thomy Phan, Fabian Ritz, Marie Kiermeier, Lenz Belzner, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, Marc Zeller, and Claudia Linnhoff-Popien. The scenario coevolution paradigm: adaptive quality assurance for adaptive systems. *International Journal on Software Tools for Technology Transfer*, pages 1–20, 2020



trolled moments of interchange of information. Eventually, the test system is to be deployed alongside the main system so that even during run-time, on-going online tests are possible [14]. This argument has been made for more classical systems as well and thus classical software test is, too, no longer restricted to a specific phase of software development. However, in the case of self-learning systems, it is important to focus on the evolution of test cases. The capabilities of the system might not grow as experienced test designers expect them to compare to systems entirely realized by human engineering effort. Thus, it is important to conceive and formalize how tests in various phases relate to each other.

Concept 2 *(System vs. Test Antagonism)* Any adaptive systems must be subject to an equally adaptive test. Overfitting is a known issue for many machine learning techniques. In software development for complex adaptive systems, it can happen on a larger scale. Any limited test suite (we expect our applications to be too complex to run a complete, exhaustive test) might induce certain unwanted biases. Ideally, once we know about the cases our system has a hard time with, we can train it specifically for these situations. For the so-hardened system, the search mechanism that gave us the hard test cases needs to come up with even harder ones to still beat the system-under-test. Employing autonomous adaptation at this stage is expected to make that arms race more immediate and faster than it is usually achieved with human developers and testers alone.

Concept 3 *(Automated Realization)* Since the realization of tasks concerning adaptive components usually means the application of a standard machine learning process, a lot of the development effort regarding certain tasks tends to shift to an earlier phase in the process model. The most developer time when applying machine learning techniques, e.g., tends to be spent on gathering information about the problem to solve and the right setup of parameters to use; the training of the learning agent then usually follows one of a few standard procedures and can run rather automatically. However, preparing and testing the component's adaptive abilities might take a lot of effort, which might occur in the design and test phase instead of the deployment phase of the system life cycle.

Concept 4 *(Artifact Abstraction)* To provide room for and exploit the system's ability to self-adapt, many artifacts produced by the engineering process tend to become more general in nature, i.e., they tend to feature more open parameters or degrees of freedom in their description. In effect, in the place of single artifacts in a classical development process, we tend to find families of artifacts or processes generating artifacts when developing a complex adaptive system. As we assume that the previously only static artifact is still included in the set of artifacts available in its place now, we call this shift "generalization" of artifacts. Following this change, many of the activities performed during development shift their targets from concrete implementations to more general artifact, i.e., when building a test suite no longer yields a series of runnable test cases but instead produces a test case generator. When this principle is broadly applied, the development activities shift toward "meta development." The developers are concerned with setting up a process able to find good solutions autonomously instead of finding the good solutions directly.

7 Scenarios

We now want to include the issue of testing adaptive systems in our formal framework. To this end, we first introduce the notion of scenarios as the basis upon which we define tests for our system. We then include that notion in our description of software development. Finally, we extend our running example with software testing.

7.1 Describing scenarios

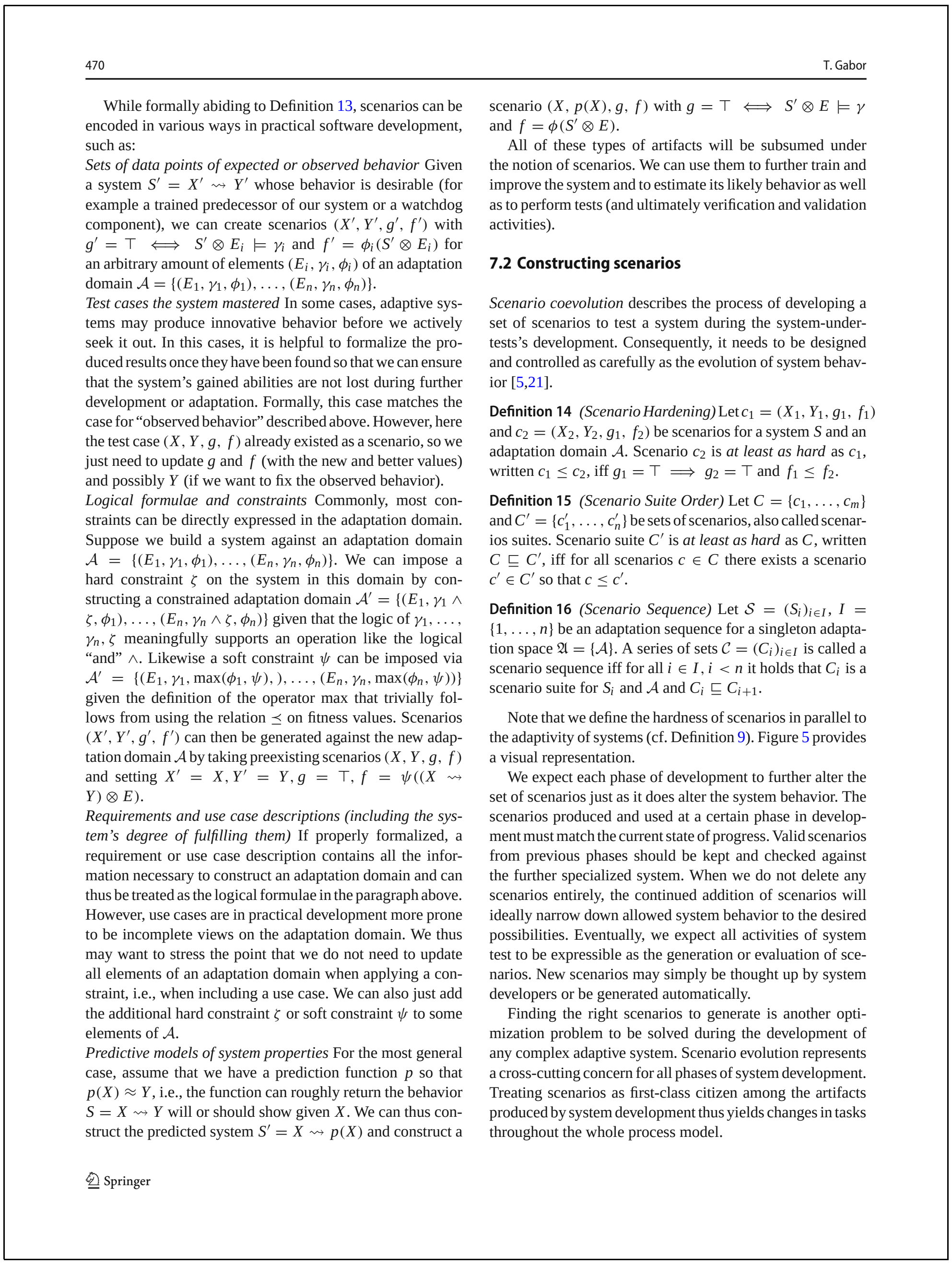
We recognize that any development process for systems following the principles described in Sect. 3 produces two central types of artifacts. The first one is a system $S = X \overset{Z}{\rightsquigarrow} Y$ with a specific desired behavior Y so that it manages to adapt to a given adaptation space. The second is a set of situations, test cases, constraints, and checked properties that this system's behavior has been validated against. We call artifacts of the second type by the group name of *scenarios*.

Definition 13 *(Scenario)* Let $S = X \overset{Z}{\rightsquigarrow} Y$ be a system and $\mathcal{A} = \{(E, \gamma, \phi)\}$ a singleton adaptation domain. A tuple $c = (X, Y, g, f)$, $g \in \{\top, \bot\}$, $f \in \mathrm{cod}(\phi)$ with $g = \top \iff S \otimes E \models \gamma$ and $f = \phi(S \otimes E)$ is called *scenario*.

Note that if we are only interested in the system's performance and not *how* it was achieved, we can redefine a scenario to leave out Y. Semantically, scenarios represent the experience that has been gained about the system's behavior during development, including both successful ($S \models \gamma$) and unsuccessful ($S \not\models \gamma$) test runs. As stated above, since we expect to operate in test spaces we cannot cover exhaustively, the knowledge about the areas we did cover is an important asset and likewise result of the systems engineering process.

Effectively, as we construct and evolve a system S, we want to construct and augment a set of scenarios $C = \{c_1, \ldots, c_n\}$ alongside with it. C is also called a *scenario suite* and can be seen as a toolbox to test S's adaptation abilities with respect to a fixed adaptation domain $\mathcal{A}$.

Taken from original publication: Thomas Gabor, Andreas Sedlmeier, Thomy Phan, Fabian Ritz, Marie Kiermeier, Lenz Belzner, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, Marc Zeller, and Claudia Linnhoff-Popien. The scenario coevolution paradigm: adaptive quality assurance for adaptive systems. *International Journal on Software Tools for Technology Transfer*, pages 1–20, 2020



While formally abiding to Definition 13, scenarios can be encoded in various ways in practical software development, such as:

Sets of data points of expected or observed behavior Given a system $S' = X' \rightsquigarrow Y'$ whose behavior is desirable (for example a trained predecessor of our system or a watchdog component), we can create scenarios (X', Y', g', f') with $g' = \top \iff S' \otimes E_i \models \gamma_i$ and $f' = \phi_i(S' \otimes E_i)$ for an arbitrary amount of elements (E_i, γ_i, ϕ_i) of an adaptation domain $\mathcal{A} = \{(E_1, \gamma_1, \phi_1), \dots, (E_n, \gamma_n, \phi_n)\}$.

Test cases the system mastered In some cases, adaptive systems may produce innovative behavior before we actively seek it out. In this cases, it is helpful to formalize the produced results once they have been found so that we can ensure that the system's gained abilities are not lost during further development or adaptation. Formally, this case matches the case for "observed behavior" described above. However, here the test case (X, Y, g, f) already existed as a scenario, so we just need to update g and f (with the new and better values) and possibly Y (if we want to fix the observed behavior).

Logical formulae and constraints Commonly, most constraints can be directly expressed in the adaptation domain. Suppose we build a system against an adaptation domain $\mathcal{A} = \{(E_1, \gamma_1, \phi_1), \dots, (E_n, \gamma_n, \phi_n)\}$. We can impose a hard constraint ζ on the system in this domain by constructing a constrained adaptation domain $\mathcal{A}' = \{(E_1, \gamma_1 \wedge \zeta, \phi_1), \dots, (E_n, \gamma_n \wedge \zeta, \phi_n)\}$ given that the logic of $\gamma_1, \dots, \gamma_n, \zeta$ meaningfully supports an operation like the logical "and" $\wedge$. Likewise a soft constraint ψ can be imposed via $\mathcal{A}' = \{(E_1, \gamma_1, \max(\phi_1, \psi),), \dots, (E_n, \gamma_n, \max(\phi_n, \psi))\}$ given the definition of the operator max that trivially follows from using the relation $\preceq$ on fitness values. Scenarios (X', Y', g', f') can then be generated against the new adaptation domain $\mathcal{A}$ by taking preexisting scenarios (X, Y, g, f) and setting $X' = X, Y' = Y, g = \top, f = \psi((X \rightsquigarrow Y) \otimes E)$.

Requirements and use case descriptions (including the system's degree of fulfilling them) If properly formalized, a requirement or use case description contains all the information necessary to construct an adaptation domain and can thus be treated as the logical formulae in the paragraph above. However, use cases are in practical development more prone to be incomplete views on the adaptation domain. We thus may want to stress the point that we do not need to update all elements of an adaptation domain when applying a constraint, i.e., when including a use case. We can also just add the additional hard constraint ζ or soft constraint ψ to some elements of $\mathcal{A}$.

Predictive models of system properties For the most general case, assume that we have a prediction function p so that $p(X) \approx Y$, i.e., the function can roughly return the behavior $S = X \rightsquigarrow Y$ will or should show given X. We can thus construct the predicted system $S' = X \rightsquigarrow p(X)$ and construct a scenario $(X, p(X), g, f)$ with $g = \top \iff S' \otimes E \models \gamma$ and $f = \phi(S' \otimes E)$.

All of these types of artifacts will be subsumed under the notion of scenarios. We can use them to further train and improve the system and to estimate its likely behavior as well as to perform tests (and ultimately verification and validation activities).

7.2 Constructing scenarios

Scenario coevolution describes the process of developing a set of scenarios to test a system during the system-under-tests's development. Consequently, it needs to be designed and controlled as carefully as the evolution of system behavior [5,21].

Definition 14 *(Scenario Hardening)* Let $c_1 = (X_1, Y_1, g_1, f_1)$ and $c_2 = (X_2, Y_2, g_1, f_2)$ be scenarios for a system S and an adaptation domain $\mathcal{A}$. Scenario c_2 is *at least as hard* as c_1, written $c_1 \leq c_2$, iff $g_1 = \top \implies g_2 = \top$ and $f_1 \leq f_2$.

Definition 15 *(Scenario Suite Order)* Let $C = \{c_1, \dots, c_m\}$ and $C' = \{c'_1, \dots, c'_n\}$ be sets of scenarios, also called scenarios suites. Scenario suite C' is *at least as hard* as C, written $C \sqsubseteq C'$, iff for all scenarios $c \in C$ there exists a scenario $c' \in C'$ so that $c \leq c'$.

Definition 16 *(Scenario Sequence)* Let $\mathcal{S} = (S_i)_{i \in I}$, $I = \{1, \dots, n\}$ be an adaptation sequence for a singleton adaptation space $\mathfrak{A} = \{\mathcal{A}\}$. A series of sets $\mathcal{C} = (C_i)_{i \in I}$ is called a scenario sequence iff for all $i \in I, i < n$ it holds that C_i is a scenario suite for S_i and $\mathcal{A}$ and $C_i \sqsubseteq C_{i+1}$.

Note that we define the hardness of scenarios in parallel to the adaptivity of systems (cf. Definition 9). Figure 5 provides a visual representation.

We expect each phase of development to further alter the set of scenarios just as it does alter the system behavior. The scenarios produced and used at a certain phase in development must match the current state of progress. Valid scenarios from previous phases should be kept and checked against the further specialized system. When we do not delete any scenarios entirely, the continued addition of scenarios will ideally narrow down allowed system behavior to the desired possibilities. Eventually, we expect all activities of system test to be expressible as the generation or evaluation of scenarios. New scenarios may simply be thought up by system developers or be generated automatically.

Finding the right scenarios to generate is another optimization problem to be solved during the development of any complex adaptive system. Scenario evolution represents a cross-cutting concern for all phases of system development. Treating scenarios as first-class citizen among the artifacts produced by system development thus yields changes in tasks throughout the whole process model.

Taken from original publication: Thomas Gabor, Andreas Sedlmeier, Thomy Phan, Fabian Ritz, Marie Kiermeier, Lenz Belzner, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, Marc Zeller, and Claudia Linnhoff-Popien. The scenario coevolution paradigm: adaptive quality assurance for adaptive systems. *International Journal on Software Tools for Technology Transfer*, pages 1–20, 2020

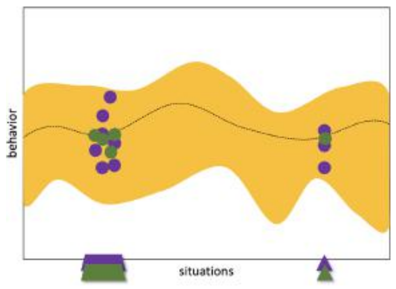


Fig. 5 Illustration of the hardness of scenarios according to Definition 14. In the same plot as in Fig. 1, scenarios from two different scenario suites C_1 (green) and C_2 (purple) can be depicted as points within the space of behavior where certain inputs need to be matched to certain outputs. Various scenario generators may cover different areas of the space of situations (shapes at the bottom of the plot). Although the depicted system $S = X \rightsquigarrow Y$ fulfills both scenario suites, C_1 is at least as hard as C_2 because its scenarios cover the same situations and require at least as close to optimal performance (colour figure online)

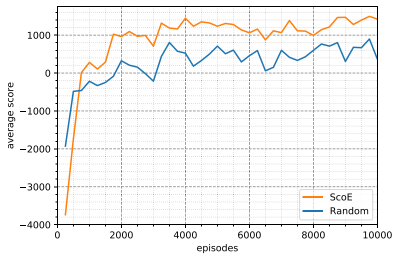


Fig. 6 Scores achieved by SCoE and standard “random” reinforcement learning during training over 10,000 episodes. Scores are averages of running the current agent against 1000 randomly generated test scenarios.. Image taken from [24] (colour figure online)

8 Example application

We now return to the *Grid World Smart Factory* domain introduced in Sect. 4. For an instance of that domain, an instance of scenario coevolution was applied in [24]. Without human involvement, a reinforcement learning agent adapting the system’s behavior and an evolutionary algorithm adapting the scenario suite have been put together. [24] has shown that the paradigm yields better results per computation time, arguing in favor of using scenario coevolution even in this fully automated form. In this section, we provide formal definition of the involved artifacts and processes fitting into the formal framework we introduced so far. We thus abstract from the dichotomy between human developers and automated adaptation and open up the paradigm of scenario coevolution to both and (most importantly) hybrid approaches.

Recall that actions in the Grid World Smart Factory as defined in Eq. 4 can be entirely simulated (although full brute force simulations of all possible actions sequences is infeasible). However, that means we can use a simulation to generate training data. And since the simulation is complete (it can simulate any situation that we defined to be able to occur within the domain), we do not need to worry about any other source of training data. In practical real-world applications, coming up with a high-fidelity simulation is usually pretty hard or expensive. Complete simulations can often be substituted with learned simulations, with are the result of machine learning themselves.

We derived the fitness function to be used in this application in Eq. 12. It allows us to steer the system toward fully producing as many items as possible. Using this fitness function, we expect the system to learn to fulfill the overall system goal of fully producing all the requested items, as defined in Eqs. 9 and 10 .

The system’s behavior is defined by the actions it chooses for each consecutive time step. In [24], we chose to program the system to execute (when in state s_i at time step i) the action

$$a_i = \max_{a \in \{a_\triangle, a_\triangleright, a_\triangledown, a_\triangleleft, a_\boxplus, a_\boxminus\}} Q(s_i, a) \tag{16}$$

where $Q(s_i, a)$ is the so-called *Q-value* of action a in state s_i. The Q-value is derived from Q-learning [41,44] and represents the expected reward when executing an action in a given state. To estimate that value, we call a neural network with weights θ.

The network weights θ are then optimized via reinforcement learning, variant of gradient descent as given in Algorithm 4. The training process runs for a fixed computational budget. For more details on the implementation in this case or any other part of the pipeline, please see [24]. For the quality of service of the trained system, we discern between the fitness function and the actual goal function. The network is trained to improve the average fitness, i.e., the average amount of items produced per run, but the user is only interested in the overall success rate, i.e., the amount of runs that are fully produced. The “random” (blue) plots in Figs. 6 and 7 show the difference: The score in Fig. 6, i.e., the value of the fitness function $\phi(S \otimes \theta)$, increases slower and on a different scale than the amount of correct runs where $S \otimes \theta \models \gamma$ shown in Fig. 7. While the network trains on the former, we assess its quality (and accept the model) using the latter.

Taken from original publication: Thomas Gabor, Andreas Sedlmeier, Thomy Phan, Fabian Ritz, Marie Kiermeier, Lenz Belzner, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, Marc Zeller, and Claudia Linnhoff-Popien. The scenario coevolution paradigm: adaptive quality assurance for adaptive systems. *International Journal on Software Tools for Technology Transfer*, pages 1–20, 2020

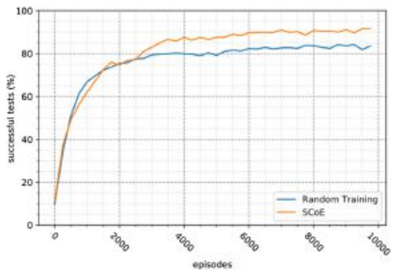


Fig. 7 Percentage of successfully solved test scenarios by SCoE and standard "random" reinforcement learning. The values are calculated from a randomly generated set of 1000 scenarios.. Image taken from [24] (colour figure online)

The learned policy is deployed on multiple independent runs. As defined in Eq. 8, these independent runs (only) differ in the position of the obstacles in the domain. For the "random" (blue) plots in Figs. 6 and 7, we generated domain instances with random obstacles. Figures 6 and 7 also show "SCoE" (orange) plots where the environments were not generated at random but by applying scenario coevolution.

For this, we need to define scenarios in the *Grid World Smart Factory* domain (also see Definition 13). We reduce complexity by not expecting specific outputs. We set no fixed requirement on goal fulfillment or fitness. The set of all possible scenarios $\mathcal{C}$ can then be given as

$$\mathcal{C} = \{(p, \emptyset, g, f) : p \in P^4, g \in \{\top, \bot\}, f \in \text{cod}(\phi)\}. \tag{17}$$

To train the system, we try to choose the hardest scenarios from $\mathcal{C}$, i.e., we try to optimize for

$$\min_{c \in \mathcal{C}} \phi(S \otimes \theta, c, \emptyset). \tag{18}$$

It is important to note that as θ changes, i.e., as the system learns, our notion for which scenarios are hard also changes.

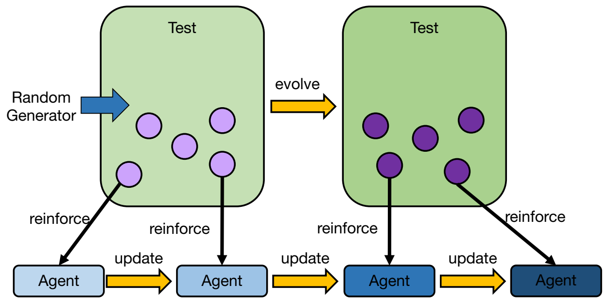
To continually come up with hard scenarios, we thus need to continually optimize for them. We do so by employing an evolutionary algorithm, which is an instance of stochastic search as given in Algorithm 1. We thus form a scenario sequence where a learning system using reinforcement and a set of scenarios generated by an evolutionary algorithm coevolve. Ideally, as the scenarios get harder, the system gets better, and as the system gets better, the scenarios get harder. Figures 6 and 7 show that employing scenario coevolution in this fully automated form already yields a benefit to the results. As discussed in [24], this benefit even upholds when considering total computational effort spent. Figure 8 depicts an overview of how the various parts of the system interact.

Within the machine learning pipeline, the search for hard scenarios represents an instantiation of the task for the specialization of the model/policy by selecting specific instances in which the model/policy is to be evaluated. However, note that while a single scenario represents a concrete domain, the whole suite of generated scenarios forms a distribution of domains and is thus an ideal artifact to use for the next iteration of the machine learning pipeline, i.e., the next generation of coevolution. Scenario coevolution as a paradigm thus instantiates the whole feedback loop constituting the machine learning pipeline.

9 Patterns for scenario coevolution

Having both introduced a formal framework for adaptation and the testing of adaptive systems using scenarios, we show

Fig. 8 Schematic representation of the scenario coevolution process for the *Grid World Smart Factory* application. A population of test scenarios is first generated at random and then improved via evolution. Between evolutions, the test scenario population is fully utilized as training data for the reinforcement learning agent, which causes the agent to improve in parallel to the test scenario population.. Image taken from [24]



Taken from original publication: Thomas Gabor, Andreas Sedlmeier, Thomy Phan, Fabian Ritz, Marie Kiermeier, Lenz Belzner, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, Marc Zeller, and Claudia Linnhoff-Popien. The scenario coevolution paradigm: adaptive quality assurance for adaptive systems. *International Journal on Software Tools for Technology Transfer*, pages 1–20, 2020

in this section how these frameworks can be applied to aid the trustworthiness of complex adaptive systems for practical use.

9.1 Criticality focus

It is very important to start the scenario evolution process alongside the system evolution, so that at each stage there exists a set of scenarios available to test the system's functionality and degree of progress (see Concept 1). This approach mimics the concept of agile development where between each sprint there exists a fully functional (however incomplete) version of the system. The concept of scenario evolution integrates seamlessly with agile process models.

In the early phases of development, the common artifacts of requirements engineering, i.e., formalized requirements, serve as the basis for the scenario evolution process. As long as the adaptation space $\mathfrak{A}$ remains constant (and with it the system goals), system development should form an adaptation sequence. Consequently, scenario evolution should then form a scenario sequence for that adaptation sequence. This means (according to Definition 16), the scenario suite is augmented with newly generated scenarios (for new system goals or just more specialized sub-goals) or with scenarios with increased requirements on fitness. Note that every change in $\mathfrak{A}$ starts new sequences. Ideally, the scenario evolution process should lead the learning components on the right path toward the desired solution. The ability to re-assign fitness priorities allows for an arms race between adaptive system and scenario suite (see Concept 2).

Augmenting requirements Beyond requirements engineering, it is necessary to include knowledge that will be generated during training and learning by the adaptive components. Mainly, recognized scenarios that work well with early version of the adaptive system should be used as checks and tests when the system becomes more complex. This approach imitates the optimization technique of importance sampling on a systems engineering level. There are two central issues that need to be answered in this early phase of the development process:

- Behavior Observation: How can system behavior be generated in a realistic manner? Are the formal specifications powerful enough? Can we employ human-labeled experience?
- Behavior Assessment: How can the quality of observed behavior be adequately assessed? Can we define a model for the users' intent? Can we employ human-labeled review?

Breaking down requirements A central task of successful requirements engineering is to split up the use cases in atomic units that ideally describe singular features. In the dynamic world, we want to leave more room for adaptive system behavior. Thus, the requirements we formulate tend to be more general in notion. It is thus even more important to split them up in meaningful ways in order to derive new sets of scenarios. The following design axes (without any claim to completeness) may be found useful to break down requirements of adaptive systems:

- Scope and Locality: Can the goal be applied/checked locally or does it involve multiple components? Which components fall into the scope of the goal? Is emergent system behavior desirable or considered harmful?
- Decomposition and Smoothness: Can internal (possibly more specific) requirements be developed? Can the overall goal be composed from a clear set of sub-goals? Can the goal function be smoothened, for example by providing intermediate goals? Can sub-goal decomposition change dynamically via adaptation or is it structurally static?
- Uncertainty and Interaction: Are all goals given with full certainty? Is it possible to reason about the relative importance of goal fulfillment for specific goals a priori? Which dynamic goals have an interface with human users or other systems?

9.2 Adaptation cool-down

We call the problem domain available to us during system design the *off-site domain*. It contains all scenarios we think the system might end up in and may thus even contain contradicting scenarios, for example. In all but the rarest cases, the situations one single instance of our system will face in its operating time will be just a fraction the size of the covered areas of the off-site domain. Nonetheless, it is also common for the system's real-world experience to include scenarios not occurring in the off-site domain at all; this mainly happens when we were wrong about some detail in the real world. Thus, the implementation of an adaptation technique faces a problem not unlike the *exploration/exploitation dilemma* [16], but on a larger scale: We need to decide, if we opt for a system fully adapted to the exact off-site domain or if we opt for a less specialized system that leaves more room for later adaptation at the customer's site. The point at which we stop adaptation happening on off-site scenarios is called the off-site adaptation border and is a key artifact of the development process for adaptive systems.

In many cases, we may want the system we build to be able to evolve beyond the exact use cases we knew about during design time. The system thus needs to have components capable of *run-time* or *online adaptation*. In the wording of this work, we also talk about *on-site adaptation* stressing that in this case we focus on adaptation processes that take place at the customer's location in a comparatively specific domain

Taken from original publication: Thomas Gabor, Andreas Sedlmeier, Thomy Phan, Fabian Ritz, Marie Kiermeier, Lenz Belzner, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, Marc Zeller, and Claudia Linnhoff-Popien. The scenario coevolution paradigm: adaptive quality assurance for adaptive systems. *International Journal on Software Tools for Technology Transfer*, pages 1–20, 2020

instead of the broader setting in a system development lab. Usually, we expect the training and optimization performed on-site (if any) to be not as drastic as training done during development. (Otherwise, we would probably have not specified our problem domain in an appropriate way.) As the system becomes more efficient in its behavior, we want to gradually reduce the amount of change we allow. In the long run, adaptation should usually work at a level that prohibits sudden, unexpected changes but still manages to handle any changes in the environment within a certain margin. The recognized need for more drastic change should usually trigger human supervision first.

Definition 17 *(Adaptation Space Sequence)* Let S be a system. A series of $|I|$ adaptation spaces $\mathbb{A} = (\mathfrak{A}_i)_{i \in I}$ with index set I with a preorder $\leq$ on the elements of I is called an *adaptation domain sequence* iff for all $i, j \in I, i \leq j$ it holds that: S adapts to $\mathfrak{A}_j$ implies that S adapts to $\mathfrak{A}_i$.

System development constructs an adaptation space sequence (cf. Concept 4), i.e., a sequence of increasingly specific adaptation domains. Each of those can be used to run an adaptation sequence (cf. Definition 10) and a scenario sequence (cf. Definition 16, Concept 2) to test it.

For the gradual reduction of the allowed amount of adaptation for the system, we use the metaphor of a "cool-down" process. The adaptation performed on-site should allow for less change than off-site adaptation. And the adaptation allowed during run-time should be less than what we allowed during deployment. This ensures that decisions that have once been deemed right by the developers are hard to change later by accident or by the autonomous adaptation process.

9.3 Eternal deployment

For high trustworthiness, development of the test cases used for the final system test should be as decoupled from the on-going scenario evolution as possible, i.e., the data used in both processes should overlap as little as possible. Of course, following this guideline completely results in the duplication of a lot of processes and artifacts. Still, it is important to accurately keep track of the influences on the respective sets of scenarios. A clear definition of the off-site adaptation border provides a starting point for when to branch off a scenario evolution process that is independent of possible scenario-specific adaptations on the system-under-test's side. Running multiple independent system tests (cf. ensemble methods [18,25]) is advisable as well. However, the space of available independently generated data is usually very limited.

For the deployment phase, it is thus of key importance to carry over as much information as possible about the genesis of the system we deploy into the run-time, where it can be used to look up the traces of observed decisions. The reason to do this now is that we usually expect the responsibility for the system to change at this point. Whereas previously, any system behavior was overseen by the developers who could potentially backtrack any phenomenon to all previous steps in the system development process, now we expect on-site maintenance to be able to handle any potential problem with the system in the real world, requiring more intricate preparation for maintenance tasks (cf. Concept 3). We thus need to endow these new people with the ability to properly understand what the system does and why.

Our approach follows the vision of *eternal system design* [33], which is a fundamental change in the way to treat deployment: We no longer ship a single artifact as the result of a complex development process, but we ship an image of the process itself (cf. Concept 4). As a natural consequence, we can only ever add to an eternal system but hardly remove changes and any trace of them entirely. Using an adequate combination operator, this meta-design pattern is already implemented in the way we construct adaptation sequences (cf. Definition 10): For example, given a system S_i we could construct $S_{i+1} = X \overset{Z}{\rightsquigarrow} Y$ in a way so that S_i is included in S_{i+1}'s internal state Z.

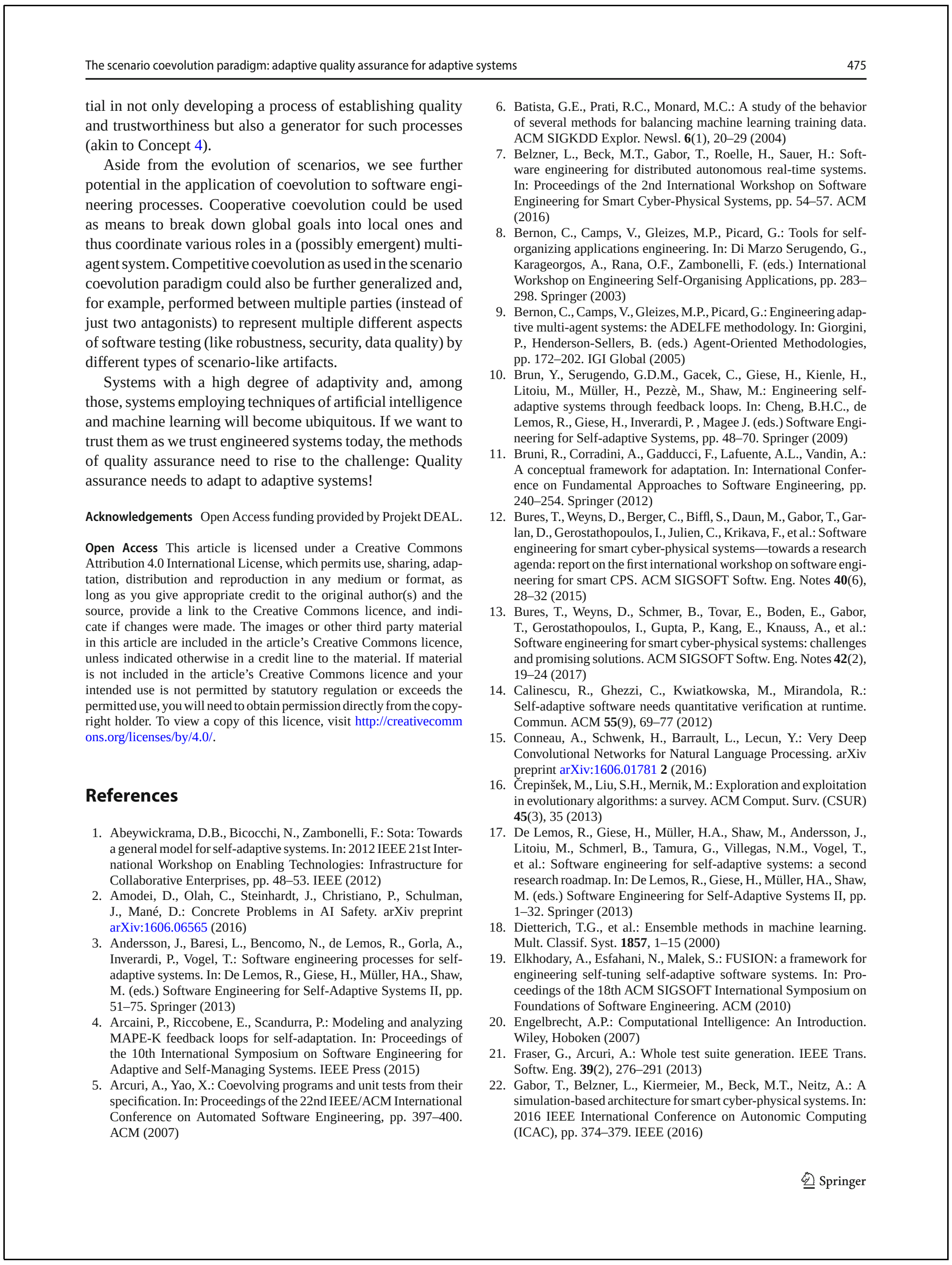
As of now, however, the design of eternal systems still raises many unanswered questions in system design. We thus resort to the notion of scenarios only as a sufficient system description to provide explanatory power at run-time and recommend to apply standard "destructive updates" to all other system artifacts.

10 Conclusion

We have introduced a new formal model for adaptation and test processes using our notion of scenarios. We connected this model to concrete challenges and arising concepts in software engineering to show that our approach of scenario coevolution is fit to tackle (a first few) of the problems when doing quality assurance for complex adaptive systems. We have put our approach into context by applying it to an example application and deriving a pipeline for the development of machine learning components from it.

As already noted throughout the text, a few challenges still persist. Perhaps most importantly, we require an adequate data structure both for the coding of systems and for the encoding of test suites and need to prove the practical feasibility of an optimization process governing the software development life cycle. For performance reasons, we expect that some restrictions on the general formal framework will be necessary. In this work, we also deliberately left out the issue of meta-processes: The software development life cycle can itself be regarded as system according to Definition 1. While this may complicate things at first, we also see poten-

Taken from original publication: Thomas Gabor, Andreas Sedlmeier, Thomy Phan, Fabian Ritz, Marie Kiermeier, Lenz Belzner, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, Marc Zeller, and Claudia Linnhoff-Popien. The scenario coevolution paradigm: adaptive quality assurance for adaptive systems. *International Journal on Software Tools for Technology Transfer*, pages 1–20, 2020



tial in not only developing a process of establishing quality and trustworthiness but also a generator for such processes (akin to Concept 4).

Aside from the evolution of scenarios, we see further potential in the application of coevolution to software engineering processes. Cooperative coevolution could be used as means to break down global goals into local ones and thus coordinate various roles in a (possibly emergent) multi-agent system. Competitive coevolution as used in the scenario coevolution paradigm could also be further generalized and, for example, performed between multiple parties (instead of just two antagonists) to represent multiple different aspects of software testing (like robustness, security, data quality) by different types of scenario-like artifacts.

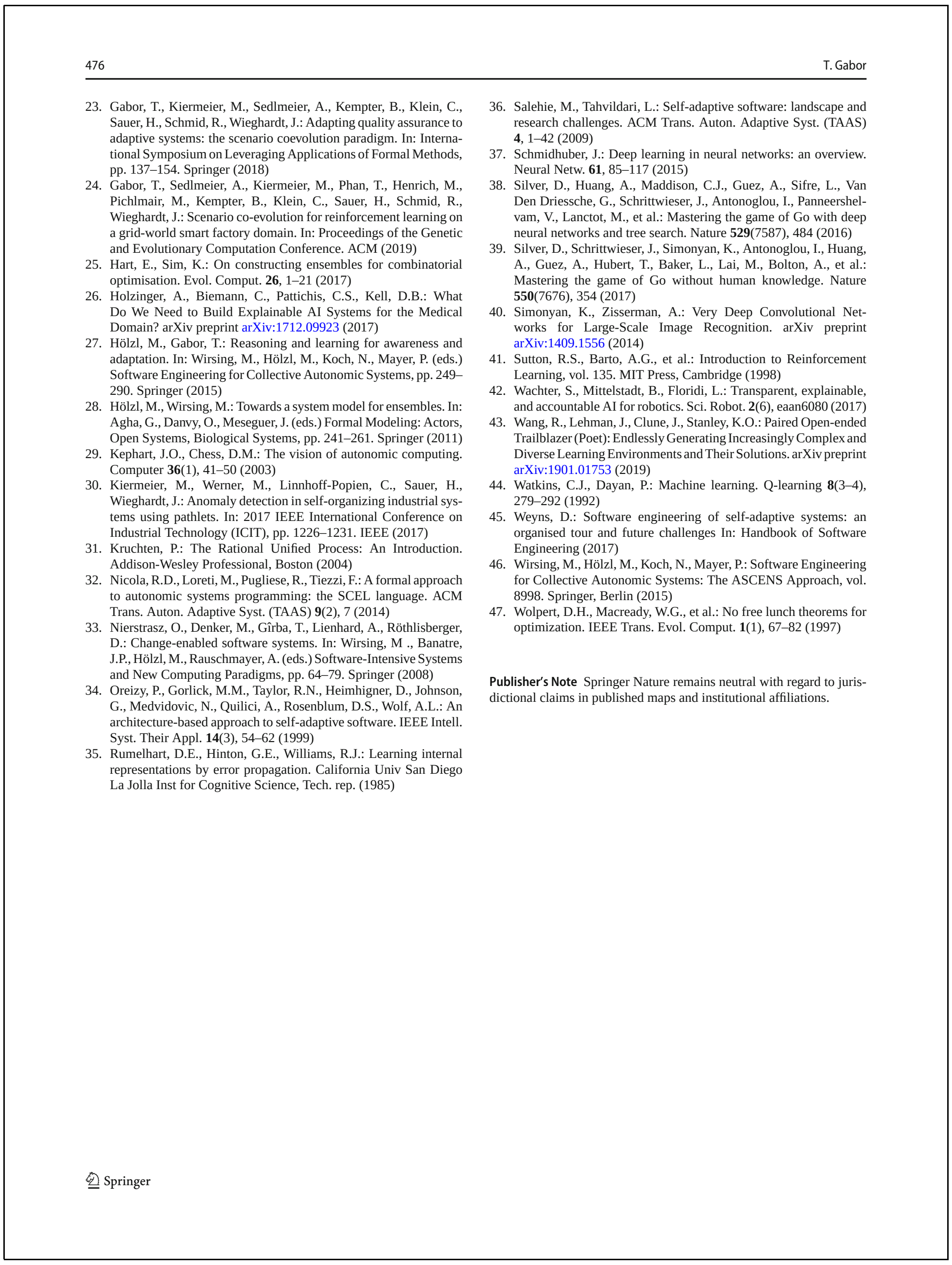
Systems with a high degree of adaptivity and, among those, systems employing techniques of artificial intelligence and machine learning will become ubiquitous. If we want to trust them as we trust engineered systems today, the methods of quality assurance need to rise to the challenge: Quality assurance needs to adapt to adaptive systems!

Acknowledgements Open Access funding provided by Projekt DEAL.

References

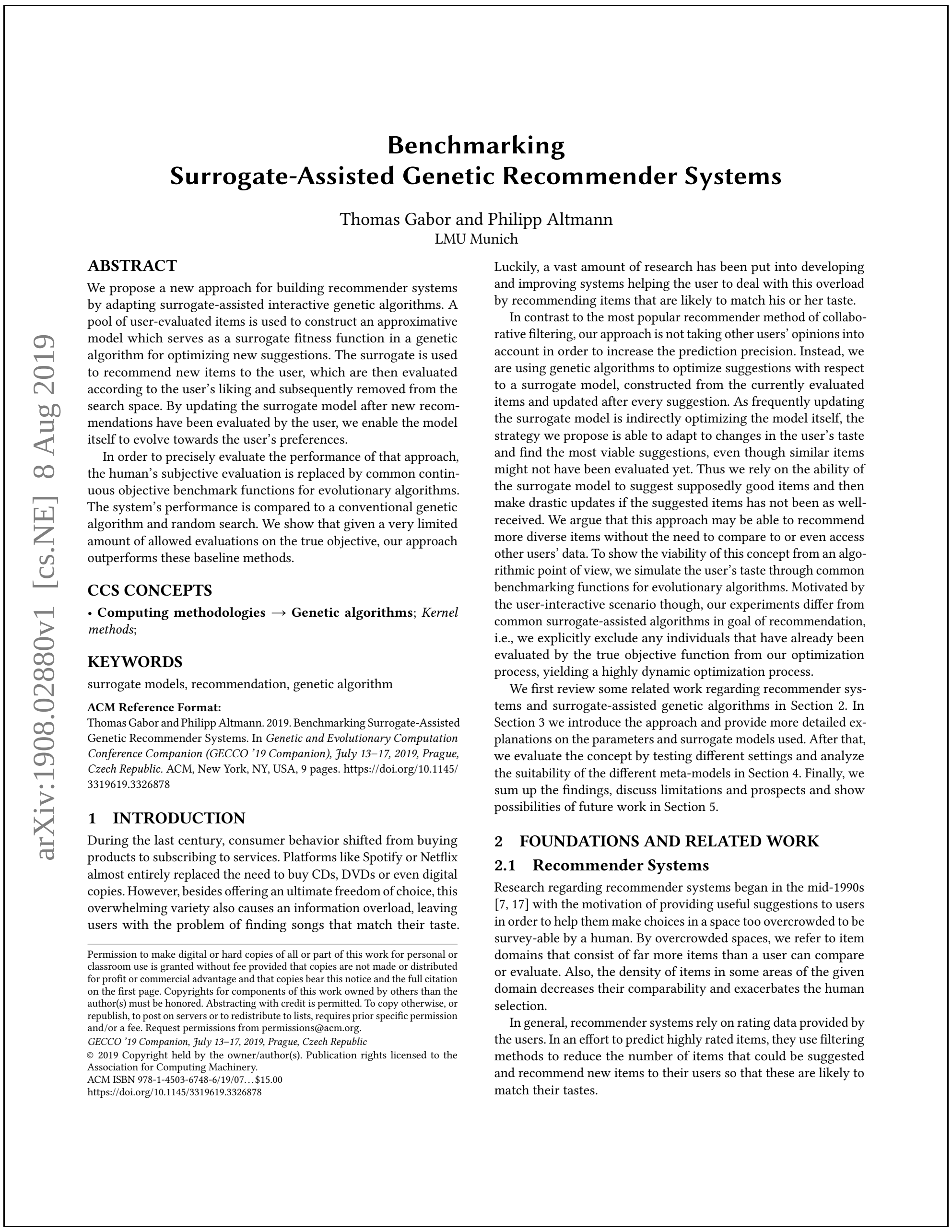
1. Abeywickrama, D.B., Bicocchi, N., Zambonelli, F.: Sota: Towards a general model for self-adaptive systems. In: 2012 IEEE 21st International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, pp. 48–53. IEEE (2012)
2. Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., Mané, D.: Concrete Problems in AI Safety. arXiv preprint arXiv:1606.06565 (2016)
3. Andersson, J., Baresi, L., Bencomo, N., de Lemos, R., Gorla, A., Inverardi, P., Vogel, T.: Software engineering processes for self-adaptive systems. In: De Lemos, R., Giese, H., Müller, HA., Shaw, M. (eds.) Software Engineering for Self-Adaptive Systems II, pp. 51–75. Springer (2013)
4. Arcaini, P., Riccobene, E., Scandurra, P.: Modeling and analyzing MAPE-K feedback loops for self-adaptation. In: Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. IEEE Press (2015)
5. Arcuri, A., Yao, X.: Coevolving programs and unit tests from their specification. In: Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering, pp. 397–400. ACM (2007)
6. Batista, G.E., Prati, R.C., Monard, M.C.: A study of the behavior of several methods for balancing machine learning training data. ACM SIGKDD Explor. Newsl. **6**(1), 20–29 (2004)
7. Belzner, L., Beck, M.T., Gabor, T., Roelle, H., Sauer, H.: Software engineering for distributed autonomous real-time systems. In: Proceedings of the 2nd International Workshop on Software Engineering for Smart Cyber-Physical Systems, pp. 54–57. ACM (2016)
8. Bernon, C., Camps, V., Gleizes, M.P., Picard, G.: Tools for self-organizing applications engineering. In: Di Marzo Serugendo, G., Karageorgos, A., Rana, O.F., Zambonelli, F. (eds.) International Workshop on Engineering Self-Organising Applications, pp. 283–298. Springer (2003)
9. Bernon, C., Camps, V., Gleizes, M.P., Picard, G.: Engineering adaptive multi-agent systems: the ADELFE methodology. In: Giorgini, P., Henderson-Sellers, B. (eds.) Agent-Oriented Methodologies, pp. 172–202. IGI Global (2005)
10. Brun, Y., Serugendo, G.D.M., Gacek, C., Giese, H., Kienle, H., Litoiu, M., Müller, H., Pezzè, M., Shaw, M.: Engineering self-adaptive systems through feedback loops. In: Cheng, B.H.C., de Lemos, R., Giese, H., Inverardi, P. , Magee J. (eds.) Software Engineering for Self-adaptive Systems, pp. 48–70. Springer (2009)
11. Bruni, R., Corradini, A., Gadducci, F., Lafuente, A.L., Vandin, A.: A conceptual framework for adaptation. In: International Conference on Fundamental Approaches to Software Engineering, pp. 240–254. Springer (2012)
12. Bures, T., Weyns, D., Berger, C., Biffl, S., Daun, M., Gabor, T., Garlan, D., Gerostathopoulos, I., Julien, C., Krikava, F., et al.: Software engineering for smart cyber-physical systems—towards a research agenda: report on the first international workshop on software engineering for smart CPS. ACM SIGSOFT Softw. Eng. Notes **40**(6), 28–32 (2015)
13. Bures, T., Weyns, D., Schmer, B., Tovar, E., Boden, E., Gabor, T., Gerostathopoulos, I., Gupta, P., Kang, E., Knauss, A., et al.: Software engineering for smart cyber-physical systems: challenges and promising solutions. ACM SIGSOFT Softw. Eng. Notes **42**(2), 19–24 (2017)
14. Calinescu, R., Ghezzi, C., Kwiatkowska, M., Mirandola, R.: Self-adaptive software needs quantitative verification at runtime. Commun. ACM **55**(9), 69–77 (2012)
15. Conneau, A., Schwenk, H., Barrault, L., Lecun, Y.: Very Deep Convolutional Networks for Natural Language Processing. arXiv preprint arXiv:1606.01781 **2** (2016)
16. Črepinšek, M., Liu, S.H., Mernik, M.: Exploration and exploitation in evolutionary algorithms: a survey. ACM Comput. Surv. (CSUR) **45**(3), 35 (2013)
17. De Lemos, R., Giese, H., Müller, H.A., Shaw, M., Andersson, J., Litoiu, M., Schmerl, B., Tamura, G., Villegas, N.M., Vogel, T., et al.: Software engineering for self-adaptive systems: a second research roadmap. In: De Lemos, R., Giese, H., Müller, HA., Shaw, M. (eds.) Software Engineering for Self-Adaptive Systems II, pp. 1–32. Springer (2013)
18. Dietterich, T.G., et al.: Ensemble methods in machine learning. Mult. Classif. Syst. **1857**, 1–15 (2000)
19. Elkhodary, A., Esfahani, N., Malek, S.: FUSION: a framework for engineering self-tuning self-adaptive software systems. In: Proceedings of the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM (2010)
20. Engelbrecht, A.P.: Computational Intelligence: An Introduction. Wiley, Hoboken (2007)
21. Fraser, G., Arcuri, A.: Whole test suite generation. IEEE Trans. Softw. Eng. **39**(2), 276–291 (2013)
22. Gabor, T., Belzner, L., Kiermeier, M., Beck, M.T., Neitz, A.: A simulation-based architecture for smart cyber-physical systems. In: 2016 IEEE International Conference on Autonomic Computing (ICAC), pp. 374–379. IEEE (2016)

Taken from original publication: Thomas Gabor, Andreas Sedlmeier, Thomy Phan, Fabian Ritz, Marie Kiermeier, Lenz Belzner, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, Marc Zeller, and Claudia Linnhoff-Popien. The scenario coevolution paradigm: adaptive quality assurance for adaptive systems. *International Journal on Software Tools for Technology Transfer*, pages 1–20, 2020



23. Gabor, T., Kiermeier, M., Sedlmeier, A., Kempter, B., Klein, C., Sauer, H., Schmid, R., Wieghardt, J.: Adapting quality assurance to adaptive systems: the scenario coevolution paradigm. In: International Symposium on Leveraging Applications of Formal Methods, pp. 137–154. Springer (2018)
24. Gabor, T., Sedlmeier, A., Kiermeier, M., Phan, T., Henrich, M., Pichlmair, M., Kempter, B., Klein, C., Sauer, H., Schmid, R., Wieghardt, J.: Scenario co-evolution for reinforcement learning on a grid-world smart factory domain. In: Proceedings of the Genetic and Evolutionary Computation Conference. ACM (2019)
25. Hart, E., Sim, K.: On constructing ensembles for combinatorial optimisation. Evol. Comput. **26**, 1–21 (2017)
26. Holzinger, A., Biemann, C., Pattichis, C.S., Kell, D.B.: What Do We Need to Build Explainable AI Systems for the Medical Domain? arXiv preprint arXiv:1712.09923 (2017)
27. Hölzl, M., Gabor, T.: Reasoning and learning for awareness and adaptation. In: Wirsing, M., Hölzl, M., Koch, N., Mayer, P. (eds.) Software Engineering for Collective Autonomic Systems, pp. 249–290. Springer (2015)
28. Hölzl, M., Wirsing, M.: Towards a system model for ensembles. In: Agha, G., Danvy, O., Meseguer, J. (eds.) Formal Modeling: Actors, Open Systems, Biological Systems, pp. 241–261. Springer (2011)
29. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer **36**(1), 41–50 (2003)
30. Kiermeier, M., Werner, M., Linnhoff-Popien, C., Sauer, H., Wieghardt, J.: Anomaly detection in self-organizing industrial systems using pathlets. In: 2017 IEEE International Conference on Industrial Technology (ICIT), pp. 1226–1231. IEEE (2017)
31. Kruchten, P.: The Rational Unified Process: An Introduction. Addison-Wesley Professional, Boston (2004)
32. Nicola, R.D., Loreti, M., Pugliese, R., Tiezzi, F.: A formal approach to autonomic systems programming: the SCEL language. ACM Trans. Auton. Adaptive Syst. (TAAS) **9**(2), 7 (2014)
33. Nierstrasz, O., Denker, M., Gîrba, T., Lienhard, A., Röthlisberger, D.: Change-enabled software systems. In: Wirsing, M ., Banatre, J.P., Hölzl, M., Rauschmayer, A. (eds.) Software-Intensive Systems and New Computing Paradigms, pp. 64–79. Springer (2008)
34. Oreizy, P., Gorlick, M.M., Taylor, R.N., Heimhigner, D., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D.S., Wolf, A.L.: An architecture-based approach to self-adaptive software. IEEE Intell. Syst. Their Appl. **14**(3), 54–62 (1999)
35. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning internal representations by error propagation. California Univ San Diego La Jolla Inst for Cognitive Science, Tech. rep. (1985)
36. Salehie, M., Tahvildari, L.: Self-adaptive software: landscape and research challenges. ACM Trans. Auton. Adapt. Syst. (TAAS) **4**, 1–42 (2009)
37. Schmidhuber, J.: Deep learning in neural networks: an overview. Neural Netw. **61**, 85–117 (2015)
38. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al.: Mastering the game of Go with deep neural networks and tree search. Nature **529**(7587), 484 (2016)
39. Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al.: Mastering the game of Go without human knowledge. Nature **550**(7676), 354 (2017)
40. Simonyan, K., Zisserman, A.: Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv preprint arXiv:1409.1556 (2014)
41. Sutton, R.S., Barto, A.G., et al.: Introduction to Reinforcement Learning, vol. 135. MIT Press, Cambridge (1998)
42. Wachter, S., Mittelstadt, B., Floridi, L.: Transparent, explainable, and accountable AI for robotics. Sci. Robot. **2**(6), eaan6080 (2017)
43. Wang, R., Lehman, J., Clune, J., Stanley, K.O.: Paired Open-ended Trailblazer (Poet): Endlessly Generating Increasingly Complex and Diverse Learning Environments and Their Solutions. arXiv preprint arXiv:1901.01753 (2019)
44. Watkins, C.J., Dayan, P.: Machine learning. Q-learning **8**(3–4), 279–292 (1992)
45. Weyns, D.: Software engineering of self-adaptive systems: an organised tour and future challenges In: Handbook of Software Engineering (2017)
46. Wirsing, M., Hölzl, M., Koch, N., Mayer, P.: Software Engineering for Collective Autonomic Systems: The ASCENS Approach, vol. 8998. Springer, Berlin (2015)
47. Wolpert, D.H., Macready, W.G., et al.: No free lunch theorems for optimization. IEEE Trans. Evol. Comput. **1**(1), 67–82 (1997)

Taken from original publication: Thomas Gabor and Philipp Altmann. Benchmarking surrogate-assisted genetic recommender systems. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1568–1575, 2019



Benchmarking Surrogate-Assisted Genetic Recommender Systems

Thomas Gabor and Philipp Altmann
LMU Munich

ABSTRACT

We propose a new approach for building recommender systems by adapting surrogate-assisted interactive genetic algorithms. A pool of user-evaluated items is used to construct an approximative model which serves as a surrogate fitness function in a genetic algorithm for optimizing new suggestions. The surrogate is used to recommend new items to the user, which are then evaluated according to the user's liking and subsequently removed from the search space. By updating the surrogate model after new recommendations have been evaluated by the user, we enable the model itself to evolve towards the user's preferences.

In order to precisely evaluate the performance of that approach, the human's subjective evaluation is replaced by common continuous objective benchmark functions for evolutionary algorithms. The system's performance is compared to a conventional genetic algorithm and random search. We show that given a very limited amount of allowed evaluations on the true objective, our approach outperforms these baseline methods.

CCS CONCEPTS

• **Computing methodologies → Genetic algorithms**; *Kernel methods*;

KEYWORDS

surrogate models, recommendation, genetic algorithm

ACM Reference Format:
Thomas Gabor and Philipp Altmann. 2019. Benchmarking Surrogate-Assisted Genetic Recommender Systems. In *Genetic and Evolutionary Computation Conference Companion (GECCO '19 Companion), July 13–17, 2019, Prague, Czech Republic.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3319619.3326878

1 INTRODUCTION

During the last century, consumer behavior shifted from buying products to subscribing to services. Platforms like Spotify or Netflix almost entirely replaced the need to buy CDs, DVDs or even digital copies. However, besides offering an ultimate freedom of choice, this overwhelming variety also causes an information overload, leaving users with the problem of finding songs that match their taste. Luckily, a vast amount of research has been put into developing and improving systems helping the user to deal with this overload by recommending items that are likely to match his or her taste.

In contrast to the most popular recommender method of collaborative filtering, our approach is not taking other users' opinions into account in order to increase the prediction precision. Instead, we are using genetic algorithms to optimize suggestions with respect to a surrogate model, constructed from the currently evaluated items and updated after every suggestion. As frequently updating the surrogate model is indirectly optimizing the model itself, the strategy we propose is able to adapt to changes in the user's taste and find the most viable suggestions, even though similar items might not have been evaluated yet. Thus we rely on the ability of the surrogate model to suggest supposedly good items and then make drastic updates if the suggested items has not been as well-received. We argue that this approach may be able to recommend more diverse items without the need to compare to or even access other users' data. To show the viability of this concept from an algorithmic point of view, we simulate the user's taste through common benchmarking functions for evolutionary algorithms. Motivated by the user-interactive scenario though, our experiments differ from common surrogate-assisted algorithms in goal of recommendation, i.e., we explicitly exclude any individuals that have already been evaluated by the true objective function from our optimization process, yielding a highly dynamic optimization process.

We first review some related work regarding recommender systems and surrogate-assisted genetic algorithms in Section 2. In Section 3 we introduce the approach and provide more detailed explanations on the parameters and surrogate models used. After that, we evaluate the concept by testing different settings and analyze the suitability of the different meta-models in Section 4. Finally, we sum up the findings, discuss limitations and prospects and show possibilities of future work in Section 5.

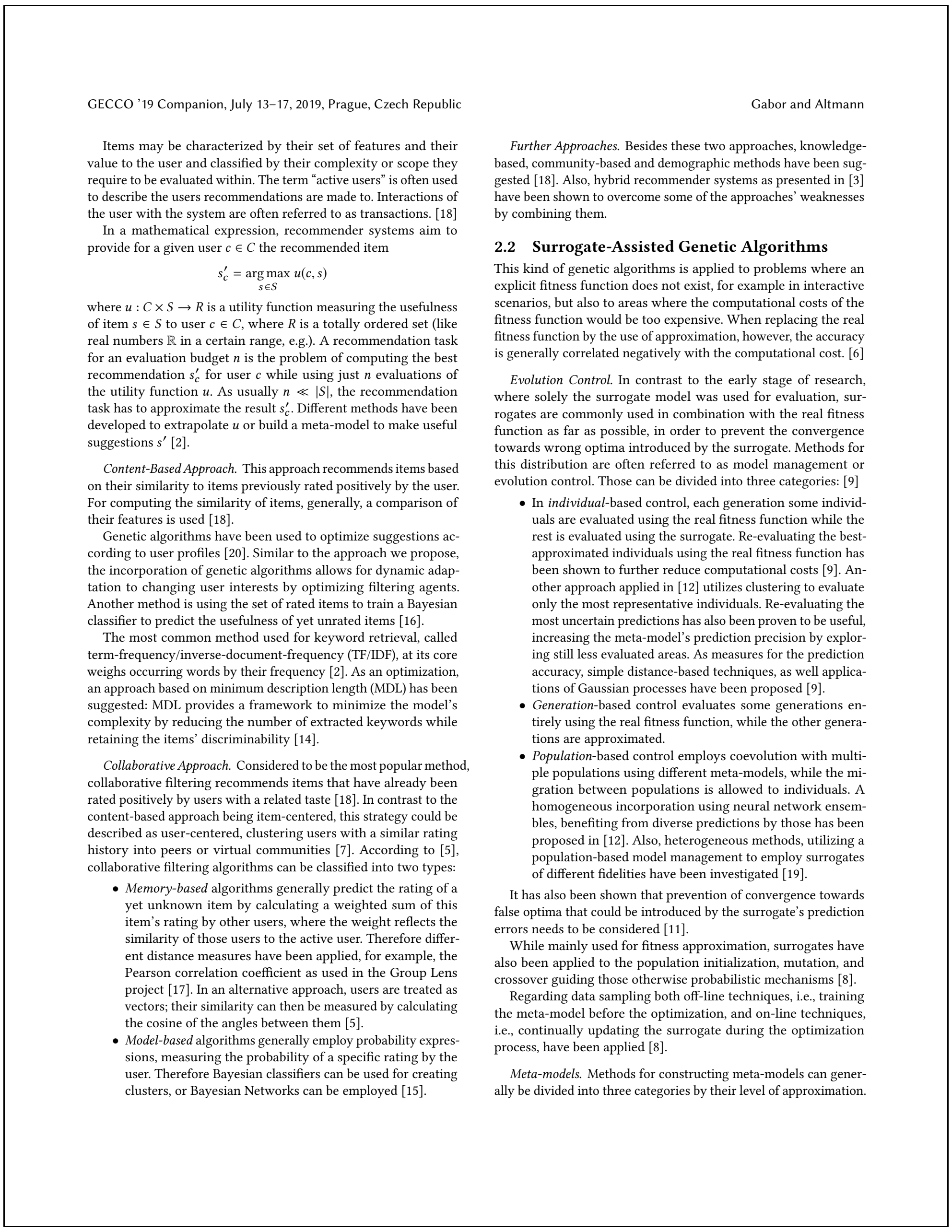
2 FOUNDATIONS AND RELATED WORK

2.1 Recommender Systems

Research regarding recommender systems began in the mid-1990s [7, 17] with the motivation of providing useful suggestions to users in order to help them make choices in a space too overcrowded to be survey-able by a human. By overcrowded spaces, we refer to item domains that consist of far more items than a user can compare or evaluate. Also, the density of items in some areas of the given domain decreases their comparability and exacerbates the human selection.

In general, recommender systems rely on rating data provided by the users. In an effort to predict highly rated items, they use filtering methods to reduce the number of items that could be suggested and recommend new items to their users so that these are likely to match their tastes.

Taken from original publication: Thomas Gabor and Philipp Altmann. Benchmarking surrogate-assisted genetic recommender systems. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1568–1575, 2019



Items may be characterized by their set of features and their value to the user and classified by their complexity or scope they require to be evaluated within. The term "active users" is often used to describe the users recommendations are made to. Interactions of the user with the system are often referred to as transactions. [18]

In a mathematical expression, recommender systems aim to provide for a given user $c \in C$ the recommended item

$$s'_c = \arg\max_{s \in S} u(c, s)$$

where $u : C \times S \longrightarrow R$ is a utility function measuring the usefulness of item $s \in S$ to user $c \in C$, where R is a totally ordered set (like real numbers $\mathbb{R}$ in a certain range, e.g.). A recommendation task for an evaluation budget n is the problem of computing the best recommendation s'_c for user c while using just n evaluations of the utility function u. As usually $n \ll |S|$, the recommendation task has to approximate the result s'_c. Different methods have been developed to extrapolate u or build a meta-model to make useful suggestions s' [2].

Content-Based Approach. This approach recommends items based on their similarity to items previously rated positively by the user. For computing the similarity of items, generally, a comparison of their features is used [18].

Genetic algorithms have been used to optimize suggestions according to user profiles [20]. Similar to the approach we propose, the incorporation of genetic algorithms allows for dynamic adaptation to changing user interests by optimizing filtering agents. Another method is using the set of rated items to train a Bayesian classifier to predict the usefulness of yet unrated items [16].

The most common method used for keyword retrieval, called term-frequency/inverse-document-frequency (TF/IDF), at its core weighs occurring words by their frequency [2]. As an optimization, an approach based on minimum description length (MDL) has been suggested: MDL provides a framework to minimize the model's complexity by reducing the number of extracted keywords while retaining the items' discriminability [14].

Collaborative Approach. Considered to be the most popular method, collaborative filtering recommends items that have already been rated positively by users with a related taste [18]. In contrast to the content-based approach being item-centered, this strategy could be described as user-centered, clustering users with a similar rating history into peers or virtual communities [7]. According to [5], collaborative filtering algorithms can be classified into two types:

- *Memory-based* algorithms generally predict the rating of a yet unknown item by calculating a weighted sum of this item's rating by other users, where the weight reflects the similarity of those users to the active user. Therefore different distance measures have been applied, for example, the Pearson correlation coefficient as used in the Group Lens project [17]. In an alternative approach, users are treated as vectors; their similarity can then be measured by calculating the cosine of the angles between them [5].
- *Model-based* algorithms generally employ probability expressions, measuring the probability of a specific rating by the user. Therefore Bayesian classifiers can be used for creating clusters, or Bayesian Networks can be employed [15].

Further Approaches. Besides these two approaches, knowledge-based, community-based and demographic methods have been suggested [18]. Also, hybrid recommender systems as presented in [3] have been shown to overcome some of the approaches' weaknesses by combining them.

2.2 Surrogate-Assisted Genetic Algorithms

This kind of genetic algorithms is applied to problems where an explicit fitness function does not exist, for example in interactive scenarios, but also to areas where the computational costs of the fitness function would be too expensive. When replacing the real fitness function by the use of approximation, however, the accuracy is generally correlated negatively with the computational cost. [6]

Evolution Control. In contrast to the early stage of research, where solely the surrogate model was used for evaluation, surrogates are commonly used in combination with the real fitness function as far as possible, in order to prevent the convergence towards wrong optima introduced by the surrogate. Methods for this distribution are often referred to as model management or evolution control. Those can be divided into three categories: [9]

- In *individual*-based control, each generation some individuals are evaluated using the real fitness function while the rest is evaluated using the surrogate. Re-evaluating the best-approximated individuals using the real fitness function has been shown to further reduce computational costs [9]. Another approach applied in [12] utilizes clustering to evaluate only the most representative individuals. Re-evaluating the most uncertain predictions has also been proven to be useful, increasing the meta-model's prediction precision by exploring still less evaluated areas. As measures for the prediction accuracy, simple distance-based techniques, as well applications of Gaussian processes have been proposed [9].
- *Generation*-based control evaluates some generations entirely using the real fitness function, while the other generations are approximated.
- *Population*-based control employs coevolution with multiple populations using different meta-models, while the migration between populations is allowed to individuals. A homogeneous incorporation using neural network ensembles, benefiting from diverse predictions by those has been proposed in [12]. Also, heterogeneous methods, utilizing a population-based model management to employ surrogates of different fidelities have been investigated [19].

It has also been shown that prevention of convergence towards false optima that could be introduced by the surrogate's prediction errors needs to be considered [11].

While mainly used for fitness approximation, surrogates have also been applied to the population initialization, mutation, and crossover guiding those otherwise probabilistic mechanisms [8].

Regarding data sampling both off-line techniques, i.e., training the meta-model before the optimization, and on-line techniques, i.e., continually updating the surrogate during the optimization process, have been applied [8].

Meta-models. Methods for constructing meta-models can generally be divided into three categories by their level of approximation.

Taken from original publication: Thomas Gabor and Philipp Altmann. Benchmarking surrogate-assisted genetic recommender systems. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1568–1575, 2019

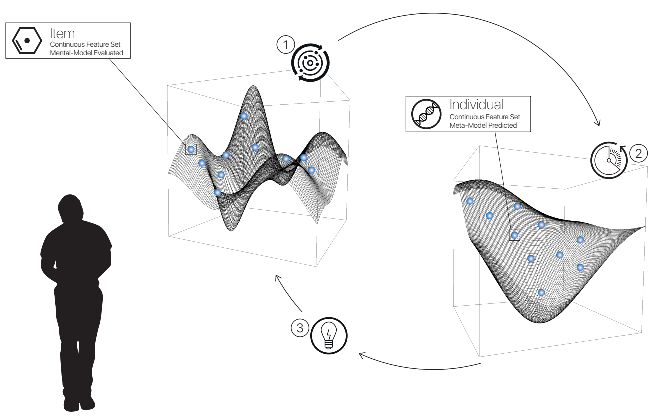


Figure 1: The surrogate-assisted genetic recommender system. A pool of user-evaluated items is used to train a meta-model (1), which in turn is used as a surrogate fitness function in a genetic algorithm optimizing recommendations (2). Once suggest-able and potentially valuable individuals are optimized, they are suggested to the user, evaluated against his or her mental-model and the meta-model is updated to optimize new suggestions (3). This cycle is carried on until the recommendations converge, i.e., no more new or valuable items are found.

Problem approximation, i.e., trying to replace the original problem with a similar problem which is easier to solve, functional approximation, i.e., trying to simplify or approximate the original fitness function, and evolutionary approximation, specific to evolutionary algorithms containing methods like fitness inheritance or the clustering strategy mentioned above [8]. As concrete approximation models, different polynomial models, Kriging models, support vector machines and neural networks have been suggested [8].

The approach presented in [13] follows our motivation and employs interactive evolutionary computation, combined with a meta-model to embed the human expert domain knowledge, solving computationally expensive modified nodal analysis.

To compensate for the lack of training data available in the use of interactive genetic algorithms in conjunction with surrogate-assisted fitness approximation, the approach presented in [21] employs co-training of radial basis function networks in a homogeneous multi surrogate fashion. Furthermore, they attempt to handle uncertainties of human interval-based fitness evaluations using a best-strategy individual-based model management.

Besides the application areas mentioned above, surrogate-assisted genetic algorithms have also been found to be helpful for dynamic optimization, constrained optimization and applied to higher optimization robustness [9, 10].

3 APPROACH

The approach we present can be classified as a content-based recommender system, using two different model-based techniques for filtering, where the results are optimized using a aurrogate-assisted genetic algorithm (SAGA). The system is built cyclically in order to improve itself among the suggestions and adapt to changes in the user's taste. See Figure 1 for visual reference depicting the system architecture as motivated by the use case of human-centered recommender systems.

For a given user $c \in C$, we assume that user c can evaluate any item $s \in S$ according to his liking. This evaluation is described by the utility function $u_c : S \rightarrow \mathbb{R}$, assuming that the user's liking can be encoded in $\mathbb{R}$. We also call u the *mental-model* of the user as it describes the user's true intent.

Note that for the purpose of this paper, we are more interested in testing the capabilities of the algorithm than testing user interaction. Thus, for the mental-model u_c we employ well-known and well-defined functions commonly used for benchmarking evolutionary algorithms, i.e., the Bohachevsky, Ackley and Schwefel functions. Accordingly, our item space is $S = \mathbb{R}^2$ to fit these functions [1].

Given a set of already evaluated items $S' \subset S$, the goal of our recommender system is to provide us with an item recommendation

$$s^* \approx \underset{s \in S \setminus S'}{\arg\max}\, u_c(s)$$

Taken from original publication: Thomas Gabor and Philipp Altmann. Benchmarking surrogate-assisted genetic recommender systems. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1568–1575, 2019

that is the best item not yet discovered by the user. However, we aim to approximate that item without actually calling u and thus allow for some error regarding optimality. Instead we return

$$\hat{s} \approx \underset{s \in S \setminus S'}{\arg\max}\ \hat{u}_c(s, S')$$

for a surrogate utility function $\hat{u}_c : S \times 2^S \rightarrow \mathbb{R}$, which we call the *meta-model*. The meta-model allows us to describe the suspected quality of items without actually evaluating them with respect to u. Note that in order to minimize errors, the meta-model may take into account all the already evaluated items S'. The defining feature of this recommendation-based instance of a SAGA is that the best recommendations according to the meta-model are then subsequently evaluated using the mental-model, i.e., $S' \leftarrow S' \cup \{\hat{s}\}$, and thus removed from the search space $S \setminus S'$.

As meta-models, a polynomial regression model fitted to the training data using the method of least squares, and an interpolation model utilizing radial basis function networks are tested and compared in this paper:

- For the surrogate model based on *polynomial regression* we employ the second order polynomial $\hat{y} = \beta_0 + \sum_{1 \leq i \leq n} \beta_i x_i + \sum_{1 \leq j \leq n} \beta_{n+j} x_j^2$ as suggested in [8]. The model is fitted to the training data, i.e., the 100 currently evaluated items using the *least squares method*, which is why we shortly refer to this surrogate as "LSM". Thus, given the $2d + 1 \times n$ input matrix X and the $n \times 1$ response matrix Y, derived from S', the fitness $u_c(s)$ for any unknown item with the d-dimensional value vector s can then be estimated by $\hat{u}_c(s, S') = s\ \hat{\Theta}$ with $\hat{\Theta} = \left(X^T X\right)^{-1} X^T Y$.
- To build an *interpolation*-based model we use *radial basis function networks* and refer to this model as "RBF". As the activation function we utilize the minimization adopted Gaussian function $\phi(x) = 1 - e^{-\left(\frac{x^2}{2\sigma^2}\right)}$.
Given a set of n input vectors and the target vector $T = (t_1, ..., t_n)$, any unknown item's fitness can be approximated by $\hat{u}_c(s, S') = \sum_{n=1}^{N} w_n \phi(\|s - s_n\|)$ with $W = \Phi^{-1} T$, where
$$\Phi = \begin{bmatrix} \phi(\|s_1 - s_1\|) & \cdots & \phi(\|s_1 - s_n\|) \\ \vdots & \ddots & \vdots \\ \phi(\|s_n - s_1\|) & \cdots & \phi(\|s_n - s_n\|) \end{bmatrix},$$
using the euclidean norm $\|p - q\|_2$ as a distance function. For finding suitable values for σ, we use a diversity-based method as suggested in [4], setting the width of all activation functions to the average distance between the currently evaluated items' values. This method results in a wider prediction landscape when the item pool is diverse and a narrower evaluation landscape once the evaluated items start converging towards an optimum.

Individuals to be recommended are selected using the best-strategy, i.e., individuals with the highest predicted fitness value according to $\hat{u}$ are recommended, thus added to the item pool S', and evaluated according to u. The amount of suggested individuals per cycle is a free parameter of this approach (cf. Section 4).

4 EVALUATION

4.1 Benchmark Objectives

In order to run a multitude of tests on the performance of the employed approach and the employed models, we opt for standard evolutionary benchmark functions instead of real human interaction. We used the implementation for Bohachevsky, Ackley, and Schwefel functions provided by [1]. See Figure 2 for a small illustration. All of these are constrained to a specific subset of $\mathbb{R}^2$ and are to be minimized with a best fitness value of 0.

4.2 Parameter Optimization

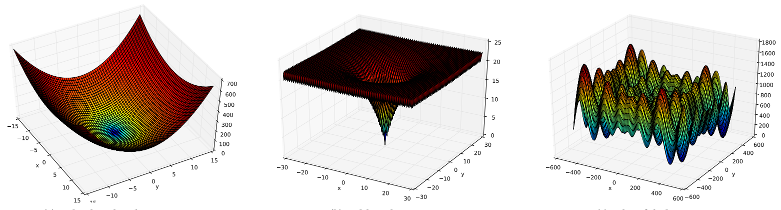
To test and compare different settings, data regarding the best item's fitness and accepted suggestion (in every suggestion cycle) as well as the fitness of the best real-evaluated item and the cycle of convergence (after the fixed number of recommendation cycles) are saved for each test run. The cycle of convergence is represented by the last recommendation cycle in which at least one suggestion was accepted. The number of accepted suggestions is obtained by counting the suggestions that are evaluated better than the worst evaluated item at that time. For enhanced comparability, the number of accepted suggestions is then normed with the number of suggestions, so that this variable displays the success in the range $[0; 1]$. The following section will provide tests and evaluations regarding the population-handling technique, the rate of evaluation, the amount of suggestions, and the optimal number of recommendation cycles. Every test setup is performed in 10 repetitions and the results are averaged in order to get a more representative result, less influenced by possible outliers, also illustrating the robustness. To keep the computational efforts reasonably low, all the tests are made based on two-dimensional versions of the objectives introduced above.

4.2.1 Rate of Evaluation and Population-Handling Technique. Test results on different evaluation rates, i.e., number of optimization cycles before suggesting, ranging from 1 to 64, as well as the impact of resetting the pool of individuals or maintaining that pool throughout the recommendations are visualized in Figure 3.

Overall, the test results imply that applying a no-reset population handling strategy yields a better outcome if the model is not at risk to converge towards a local optimum of the benchmark (see results for the Bohachevsky benchmark). A counterexample for this can be observed at the results for the LSM model on the Ackley benchmark. Also, choosing lower rates, i.e., shorter optimization of suggestions, further reduces the risk of introducing false optima to the surrogate by decelerating the model's convergence. Otherwise, higher rates are able to benefit the system's performance, as seen at the rest results for the LSM model on the Bohachevsky benchmark.

4.2.2 Amount of Suggested Individuals per Cycle. Regarding the amount of recommendations per cycle, numbers from 1 through 8 have been tested; test results can be seen in Figure 4. Overall, a higher number of suggestions benefits the systems performance, especially in combination with shorter optimization of those, as this leads to a higher diversity of recommendation, which could counteract the risk of the model's convergence towards a local or false optimum. Also, the comparably local perspective of the RBF model, causing an overall worse performance than the LSM model, seems to be neutralized by this effect, as seen in the results for the

Taken from original publication: Thomas Gabor and Philipp Altmann. Benchmarking surrogate-assisted genetic recommender systems. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1568–1575, 2019



(a) Bohachevsky objective (b) Ackley objective (c) Schwefel objective

Figure 2: Function plots for the (a) Bohachevsky, (b) Ackley, and (c) Schwefel benchmark functions for two-dimensional input. Images taken from [1].

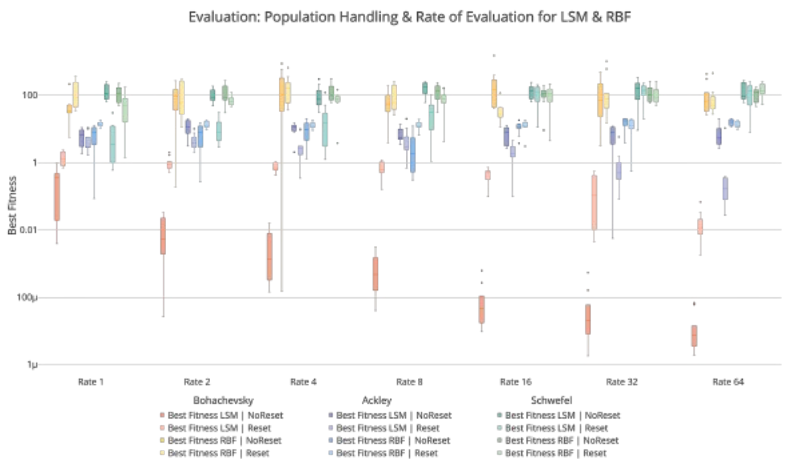


Figure 3: Benchmarking different Evaluation Rates (x-axis) on the Bohachevsky (red/orange), Ackley (blue) and Schwefel (green) objective. The LSM- and RBF-Model are compared with a Reset (lighter color) and NoReset (darker color) population-handling technique. Test results for ten iterations of each test setup are displayed by box-plots for the Best Fitness on a logarithmic scale.

RBF model on the Bohachevsky benchmark. The results for the LSM model on both the Bohachevsky and Ackley benchmarks show less correlation, as the results are already quite good and further improvement would be hard to achieve, especially with this more globally oriented surrogate model.

4.2.3 System Convergence and Number of Recommendation Cycles. As mentioned above, the tested system is intended to work in an interactive scenario; thus we strive for an overall low amount of real fitness evaluations. In order to further reduce those, additional tests have been carried out to determine the minimal amount of recommendation cycles required. The convergence of the system, i.e., if it is still able to make valuable recommendations is measured

Taken from original publication: Thomas Gabor and Philipp Altmann. Benchmarking surrogate-assisted genetic recommender systems. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1568–1575, 2019

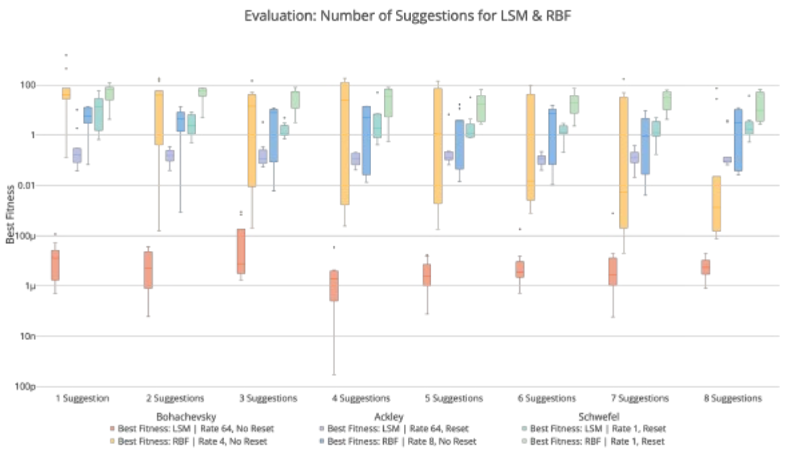


Figure 4: Testing the impact of the number of suggestions (x-axis) per recommendation cycle on the Bohachevsky (red/orange), Ackley (blue) and Schwefel (green) objective. Comparing the LSM and RBF models by their Best Fitness, displayed by box-plots on a logarithmic scale. Concrete parameter settings for the Evaluation Rate and the population-handling technique are shown in the legend below.

by the number of accepted suggestions while the performance of the surrogate and its convergence can be derived from the fitness of the currently best real-evaluated item. The test results visualized in Figure 5 show that all of the systems mostly converge within about 100 cycles, which, depending on the specific settings requires about 1000 real fitness evaluations at maximum. Reaching further improvement with a higher number of recommendation cycles could not be justified by the amount of additional real evaluations that would be needed.

4.3 Comparison of System Performance

In order to put the Surrogate-Assisted Genetic Recommender System's ("SAGRS") performance into a context, we compare it to the conventional Genetic Algorithm ("GA"). For all evolutionary processes (both within the SAGRS and for the GA itself), a selection factor of 0.9, a mutation probability of 0.1, and a recombination probability of 0.05 are used. Furthermore an equal amount of true fitness evaluations n_{eval} is retained, defining the population size n_{pop} and the number of generations n_{gen} in even distribution, such that $n_{pop} = n_{gen} = \lfloor \sqrt{n_{eval}} \rfloor$. If compared to LSM and RBF models with different amounts of true evaluations, the higher one is used to compute the settings.

To further validate the necessity and the performance of the meta-models used and to get an impression of the impact the genetic optimization has, comparisons against a random search strategy (Random Recommender, "RR"), integrated into the system the same way the genetic optimization is, are drawn. Random search is implemented as a genetic optimization with an evaluation rate of 0, causing the optimization to be skipped and the system to recommend the initially best-estimated individuals. As parameters for this random-search-adopted recommender, an evaluation rate of 0 and a reset population-handling technique as well as the optimal settings for the number of suggestions and the amount of recommendation cycles (as evaluated in the previous sub-section) are used. From a broader perspective, this system could also be seen as a conventional content-based recommender-system, making suggestions solely based on the estimation of the unknown item's rating. Concrete settings for all of those variable parameters of each system compared are annotated in the legends of the plots.

Bohachevsky. The comparison results seen in Figure 6 show that the SAGRS outperforms the Genetic Algorithm as well as the Random Recommender with both approximation models easily. The reason for this is most likely that the Genetic Algorithm cannot handle such a low amount of evaluations, i.e., it is not able to offer convergence towards better fitness areas with such small amounts of individuals and optimization cycles.

Taken from original publication: Thomas Gabor and Philipp Altmann. Benchmarking surrogate-assisted genetic recommender systems. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1568–1575, 2019

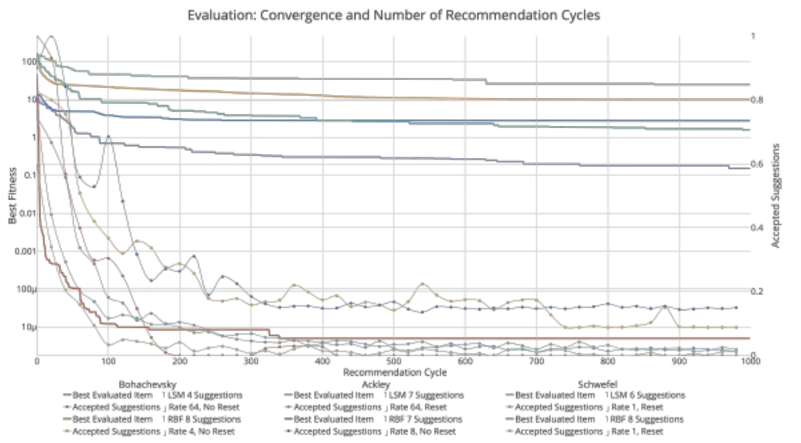


Figure 5: Evaluating the optimal number of recommendation cycles (x-axis) needed for both LSM and RBF models on the Bohachevsky (red/orange), Ackley (blue) and Schwefel (green) objective. The optimization progress represented by the Best Fitness (left y-axis) is visualized by a line graph on a logarithmic scale, the share in accepted suggestions (right y-axis) represented by dotted line graphs. Concrete parameter settings for the Evaluation Rate, population-handling technique, and number of suggested items per recommendation cycle are displayed in the legend below.

The Random Recommender, also performing better than the Genetic Algorithm, shows that the incorporation of meta-models has a distinctly positive impact on the system's performance. Comparing the Surrogate-Assisted Genetic Recommender to the Random Recommender furthermore provides evidence that the optimization of suggestions has an affirmative effect.

The LSM model apparently has the best outcome, but also the RBF model performs surprisingly well. With 100 recommendation cycles of four suggestions, optimized comparably long from a population that is retained over the runtime, the LSM model clearly yields profit from its similarity to the objective and its global estimation capabilities. The RBF model, on the other hand, takes advantage from an increased diversity of suggestions, as a result to their short optimization in conjunction with the high amount of recommendations per cycle, allowing for a better exploration of the objective, thus counteracting the model's local perspective.

Ackley. The comparison results seen in Figure 7 show that, similar to the Bohachevksy objective, the SAGRS is able to outperform the Genetic Algorithm as well as the Random Recommender with both approximation models. Still, the Genetic Algorithm does not accomplish convergence towards any good solutions with this low amount of fitness evaluations, which seems obvious, given the fact, that Ackley is even harder to be optimized than Bohachevsky, where it already failed.

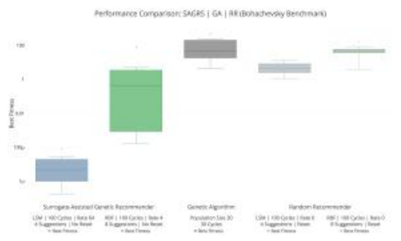


Figure 6: Performance comparison of the SAGRS using an LSM model (blue) and an RBF model (green), a conventional genetic algorithm (dark grey), and a random recommender based on the LSM (blue-grey) and RBF (green-grey) model on the Bohachevsky objective. Their outcome regarding the best fitness is visualized by box-plots on a logarithmic scale. Concrete parameter settings are shown in the legend.

Taken from original publication: Thomas Gabor and Philipp Altmann. Benchmarking surrogate-assisted genetic recommender systems. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1568–1575, 2019

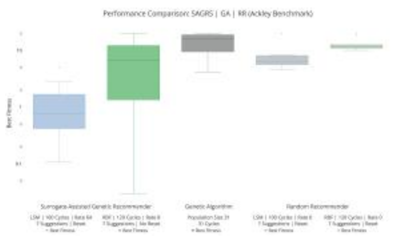


Figure 7: Performance comparison of the SAGRS using an LSM model (blue) and an RBF model (green), a conventional genetic algorithm (dark grey), and a random recommender based on the LSM (blue-grey) and RBF (green-grey) model on the Ackley objective. Their outcome regarding the best fitness is visualized by box-plots on a logarithmic scale. Concrete parameter settings are shown in the legend.

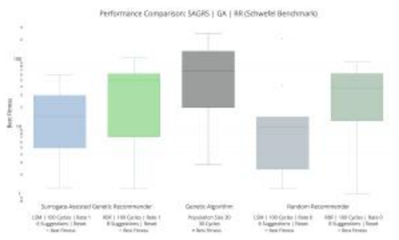


Figure 8: Performance comparison of the SAGRS using an LSM model (blue) and an RBF model (green), a conventional genetic algorithm (dark grey), and a random recommender based on the LSM (blue-grey) and RBF (green-grey) model on the Schwefel objective. Their outcome regarding the best fitness is visualized by box-plots on a logarithmic scale. Concrete parameter settings are shown in the legend.

Even though the results are allocated on a denser expanse, due to the objective's comparably small fitness range, it still can be perceived that the Random Recommender results in a better outcome than the Genetic Algorithm, while being outperformed by the SAGRS. Therefore, the implications about the usefulness of both the optimization of suggestions and the utilization of approximation models can still be endorsed for this specific benchmark.

While mostly performing worse than the LSM model, the RBF model is able to reach even better results at some times. With a comparably short optimization and a high number of suggestions, the RBF model can benefit from a higher diversity of recommendations, as seen before. In contrast, the LSM model is able to prevent being misdirected by the objective's local optima, by not retaining previous populations, therefore requiring longer optimizations, and suggesting a high amount of individuals.

Schwefel. The comparison results seen in Figure 8 show that the SAGRS is able to slightly outperform the Genetic Algorithm and offers similarly good results compared to the Random Recommender. After performing badly for the previous two benchmarks, which could be classified as easier than Schwefel, the Genetic Algorithm shows good results despite the low number of evaluations.

Here, the Random Recommender shows a comparably better performance than the SAGRS. Considering their low evaluation rate of 1, hardly differing in effect to a rate of 0, and that they the same population-handling technique, such outcome could have been presumed with regard to the test results for the Schwefel objective in the previous section. Still, the positive impact of the use of approximation models as well as the better performance of the LSM model due to its global estimation capabilities can be seen.

The short optimization, combined with the high amount of suggestions and the reinitialization of the population, causes the convergence of both meta-models to be decelerated to a high degree, preventing them from converging towards the distinct local optima of the objective.

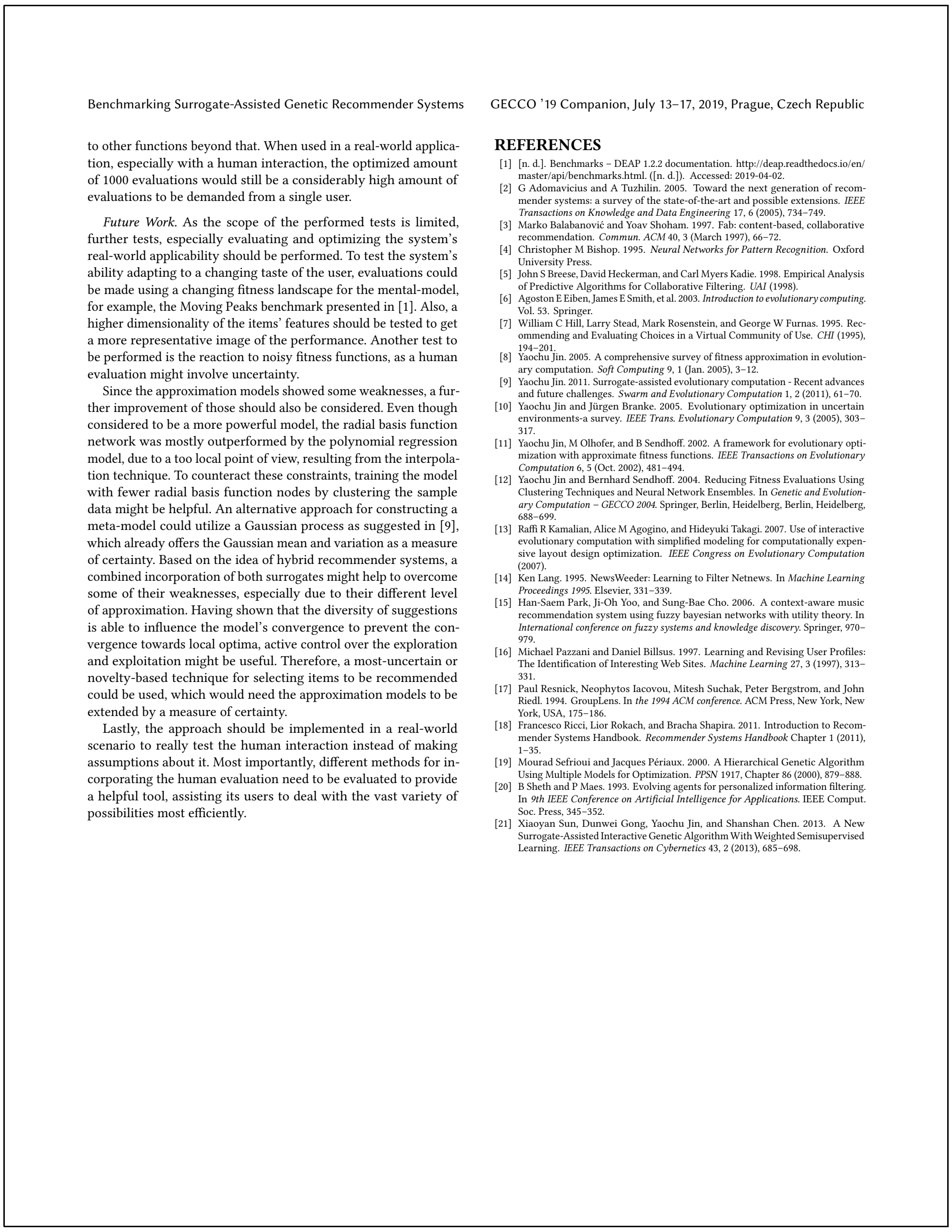
5 CONCLUSION

We proposed a framework for building systems that are able to make recommendations based on content that has been evaluated by a user. We assumed that user preference can be modeled by a real-valued function called the mental-model, i.e., the true utility to be measured from the user evaluating a given item. In order to estimate the true utility of items yet unknown to the user, we used a polynomial regression model (fitted using the method of least squares) and an interpolation model (using radial basis function networks) as surrogate models. These were employed by a genetic algorithm to optimize for the best item that has not yet been evaluated according to the mental-model. By evaluating these suggested items on the mental-model, we improve our surrogate model but also force a dynamic change in the mental-model.

We evaluated and tested the approach by replacing the subjective human evaluation with three different objective benchmark functions. Evaluating and optimizing the impact of the evaluation rate, population-handling technique, number of suggestions and number of recommendation cycles, we realized that decelerating the meta-model's convergence helps to overcome local optima of the objective, and aids the system to converge towards the global optimum.

Limitations. As we replaced the human evaluation with benchmark functions, the influences of human evaluation, even though considered, are not tested nor evaluated. Therefore an applicability of this approach as an interactive system cannot be stated. Furthermore, the use of those benchmarks compensates for the need of an appropriate classification of items, which plays an important role when applying the system to real items. Also, all tests were only made with two-dimensional values, which would be too few for accurately classifying actual items. It should also be noted that while the benchmark functions are well-established for testing genetic algorithms, it is still to be shown if the approach generalizes

Taken from original publication: Thomas Gabor and Philipp Altmann. Benchmarking surrogate-assisted genetic recommender systems. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1568–1575, 2019



to other functions beyond that. When used in a real-world application, especially with a human interaction, the optimized amount of 1000 evaluations would still be a considerably high amount of evaluations to be demanded from a single user.

Future Work. As the scope of the performed tests is limited, further tests, especially evaluating and optimizing the system's real-world applicability should be performed. To test the system's ability adapting to a changing taste of the user, evaluations could be made using a changing fitness landscape for the mental-model, for example, the Moving Peaks benchmark presented in [1]. Also, a higher dimensionality of the items' features should be tested to get a more representative image of the performance. Another test to be performed is the reaction to noisy fitness functions, as a human evaluation might involve uncertainty.

Since the approximation models showed some weaknesses, a further improvement of those should also be considered. Even though considered to be a more powerful model, the radial basis function network was mostly outperformed by the polynomial regression model, due to a too local point of view, resulting from the interpolation technique. To counteract these constraints, training the model with fewer radial basis function nodes by clustering the sample data might be helpful. An alternative approach for constructing a meta-model could utilize a Gaussian process as suggested in [9], which already offers the Gaussian mean and variation as a measure of certainty. Based on the idea of hybrid recommender systems, a combined incorporation of both surrogates might help to overcome some of their weaknesses, especially due to their different level of approximation. Having shown that the diversity of suggestions is able to influence the model's convergence to prevent the convergence towards local optima, active control over the exploration and exploitation might be useful. Therefore, a most-uncertain or novelty-based technique for selecting items to be recommended could be used, which would need the approximation models to be extended by a measure of certainty.

Lastly, the approach should be implemented in a real-world scenario to really test the human interaction instead of making assumptions about it. Most importantly, different methods for incorporating the human evaluation need to be evaluated to provide a helpful tool, assisting its users to deal with the vast variety of possibilities most efficiently.

REFERENCES

[1] [n. d.]. Benchmarks – DEAP 1.2.2 documentation. http://deap.readthedocs.io/en/master/api/benchmarks.html. ([n. d.]). Accessed: 2019-04-02.

[2] G Adomavicius and A Tuzhilin. 2005. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering* 17, 6 (2005), 734–749.

[3] Marko Balabanović and Yoav Shoham. 1997. Fab: content-based, collaborative recommendation. *Commun. ACM* 40, 3 (March 1997), 66–72.

[4] Christopher M Bishop. 1995. *Neural Networks for Pattern Recognition.* Oxford University Press.

[5] John S Breese, David Heckerman, and Carl Myers Kadie. 1998. Empirical Analysis of Predictive Algorithms for Collaborative Filtering. *UAI* (1998).

[6] Agoston E Eiben, James E Smith, et al. 2003. *Introduction to evolutionary computing.* Vol. 53. Springer.

[7] William C Hill, Larry Stead, Mark Rosenstein, and George W Furnas. 1995. Recommending and Evaluating Choices in a Virtual Community of Use. *CHI* (1995), 194–201.

[8] Yaochu Jin. 2005. A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing* 9, 1 (Jan. 2005), 3–12.

[9] Yaochu Jin. 2011. Surrogate-assisted evolutionary computation - Recent advances and future challenges. *Swarm and Evolutionary Computation* 1, 2 (2011), 61–70.

[10] Yaochu Jin and Jürgen Branke. 2005. Evolutionary optimization in uncertain environments-a survey. *IEEE Trans. Evolutionary Computation* 9, 3 (2005), 303–317.

[11] Yaochu Jin, M Olhofer, and B Sendhoff. 2002. A framework for evolutionary optimization with approximate fitness functions. *IEEE Transactions on Evolutionary Computation* 6, 5 (Oct. 2002), 481–494.

[12] Yaochu Jin and Bernhard Sendhoff. 2004. Reducing Fitness Evaluations Using Clustering Techniques and Neural Network Ensembles. In *Genetic and Evolutionary Computation – GECCO 2004.* Springer, Berlin, Heidelberg, Berlin, Heidelberg, 688–699.

[13] Raffi R Kamalian, Alice M Agogino, and Hideyuki Takagi. 2007. Use of interactive evolutionary computation with simplified modeling for computationally expensive layout design optimization. *IEEE Congress on Evolutionary Computation* (2007).

[14] Ken Lang. 1995. NewsWeeder: Learning to Filter Netnews. In *Machine Learning Proceedings 1995.* Elsevier, 331–339.

[15] Han-Saem Park, Ji-Oh Yoo, and Sung-Bae Cho. 2006. A context-aware music recommendation system using fuzzy bayesian networks with utility theory. In *International conference on fuzzy systems and knowledge discovery.* Springer, 970–979.

[16] Michael Pazzani and Daniel Billsus. 1997. Learning and Revising User Profiles: The Identification of Interesting Web Sites. *Machine Learning* 27, 3 (1997), 313–331.

[17] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. 1994. GroupLens. In *the 1994 ACM conference.* ACM Press, New York, New York, USA, 175–186.

[18] Francesco Ricci, Lior Rokach, and Bracha Shapira. 2011. Introduction to Recommender Systems Handbook. *Recommender Systems Handbook* Chapter 1 (2011), 1–35.

[19] Mourad Sefrioui and Jacques Périaux. 2000. A Hierarchical Genetic Algorithm Using Multiple Models for Optimization. *PPSN* 1917, Chapter 86 (2000), 879–888.

[20] B Sheth and P Maes. 1993. Evolving agents for personalized information filtering. In *9th IEEE Conference on Artificial Intelligence for Applications.* IEEE Comput. Soc. Press, 345–352.

[21] Xiaoyan Sun, Dunwei Gong, Yaochu Jin, and Shanshan Chen. 2013. A New Surrogate-Assisted Interactive Genetic Algorithm With Weighted Semisupervised Learning. *IEEE Transactions on Cybernetics* 43, 2 (2013), 685–698.

Taken from original publication: André Reichstaller, Thomas Gabor, and Alexander Knapp. Mutation-based test suite evolution for self-organizing systems. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2018

Mutation-based Test Suite Evolution for Self-Organizing Systems [⋆]

André Reichstaller[1], Thomas Gabor[2], and Alexander Knapp[1]

[1] Institute for Software & Systems Engineering
University of Augsburg, Germany
$\{lastname\}$@isse.de

[2] Institute for Informatics, LMU Munich, Germany
thomas.gabor@ifi.lmu.de

Abstract. We consider test design as an optimization problem. The challenge is to find a set of test cases, the so-called test suite, that optimizes two quantifiable requirements: First, the effort needed for test execution should be minimal; a given test budget usually sets a maximum for the size of the test suite. Second, the test suite should maximize the score of a given test goal estimating its error detection capability, e.g., by the use of coverage or risk metrics. This paper studies test design for testing self-organizing systems with a mutation-based test goal. Equipped with a reconfiguration mechanism, this kind of a distributed system adapts its internal structure and thus its behavior to changing environmental conditions at run time. Test execution at a time step t consequently not only triggers an observable output at $t+1$, but might also bring about a reconfiguration of the system under test influencing the result of subsequently executed test cases. Formalizing the evolving sequential decision problem of test case executions by dependency graphs, in which we try to find optimal sets of paths for the mutation-based goal, we investigate the suitability of various kinds of evolutionary algorithms for optimization. All of the considered algorithms are evaluated using a concrete case study of an adaptive, self-organizing production cell.

1 Testing Self-Organizing Systems

"Testing is the process of executing a program with the intent of finding errors" [21]. Since exhaustively executing a program with all imaginable inputs I is in practice not feasible, it is up to the tester to choose a subset $I' \subset I$ which is expected to find most of the errors. This challenge, generally referenced as *test design*, can be seen as a problem of optimization: Given a goal function $\Gamma : 2^I \to \mathbb{R}$ that quantifies the expectation of detected errors for all possible subsets of inputs, we strive to find the optimal $I' \subset I$ with $|I'| = k$, where k denotes the maximum number of permitted executions, that fulfills the constraints in time and cost. Since general solutions for this subset selection problem are computationally expensive – exhaustive search would need $\binom{|I|}{k}$ goal evaluations – it is common to solve specific instances with heuristic approaches that build on domain-specific knowledge.

[⋆] Accepted for ISoLA 2018.

Taken from original publication: André Reichstaller, Thomas Gabor, and Alexander Knapp. Mutation-based test suite evolution for self-organizing systems. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2018

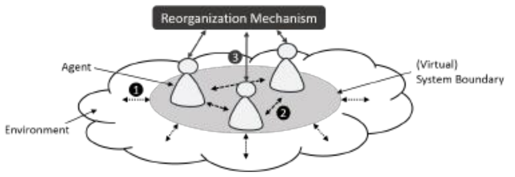


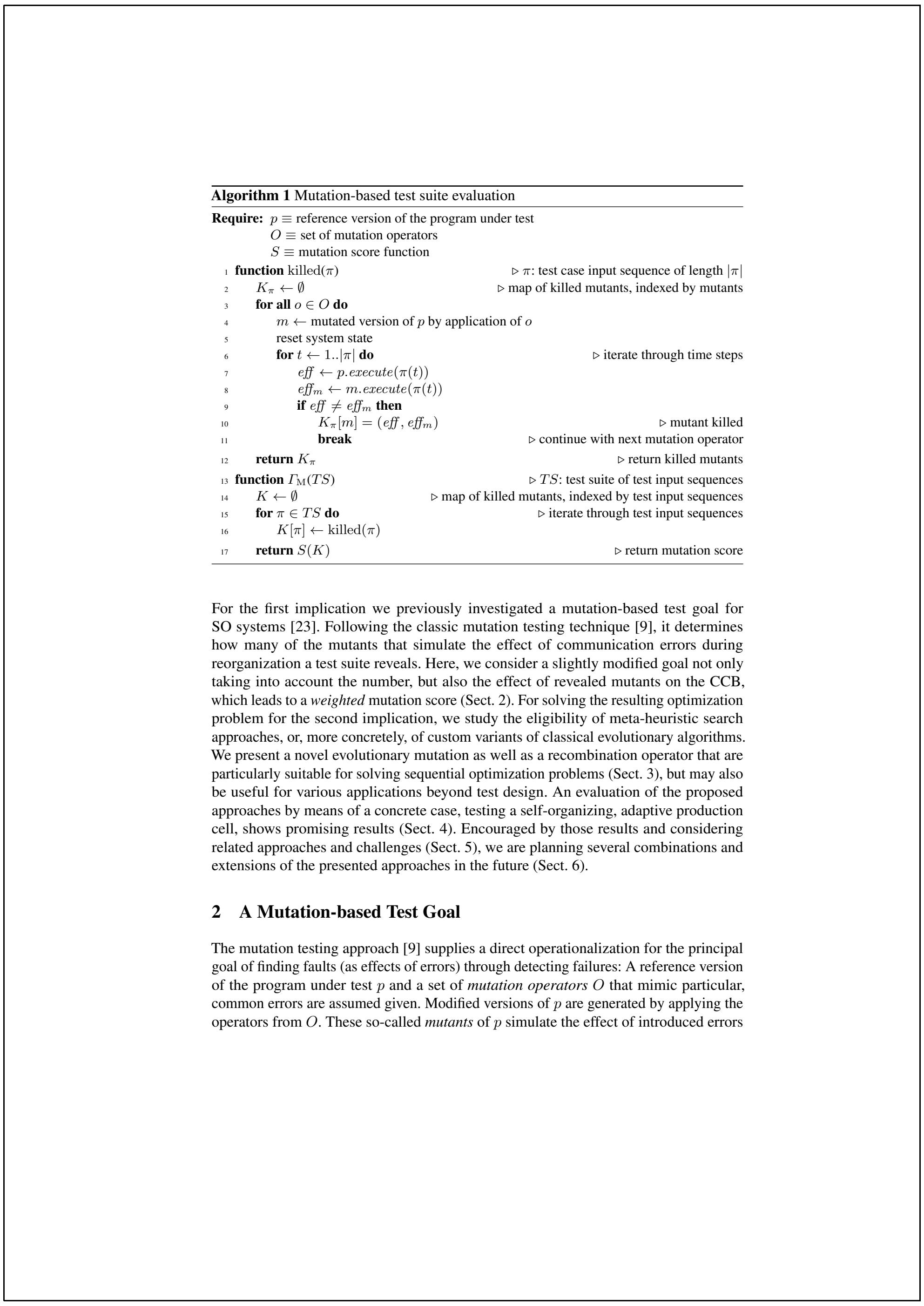
Fig. 1. Fundamental setup of self-organizing systems: Distributed software agents (often embedded in physical machines) continually interact with their environment (denoted by arrow type ❶). The joint system behavior is strongly influenced by inter-agent communication (❷) through message passing. If an agent detects an environmental fault hindering the current system approach, it triggers a reorganization mechanism (❸) which computes and distributes a new valid configuration.

Self-organizing (SO) systems [8] are distributed systems with the particular characteristic that they are able to adapt their internal structure at run time to changing environmental conditions; see Fig. 1 for the fundamental setup. This kind of self-adaptation builds on an internal system state which is spread over the physically distributed agent components and, when necessary, is modified by a so-called *reorganization mechanism* computing and distributing an adapted agent component configuration in a central or again distributed manner by message passing. Reorganizations are triggered by monitoring the current configuration and the environment. In the *Restore Invariant Approach* (RIA) [17] a "corridor of correct behavior" (CCB) is used which is described by invariants and where imminent leaving of the corridor results in a reorganization.

Though affecting the actual run-time behavior, the internal states and configuration of an SO system usually are not accessible from outside the system boundaries; changes to the system configuration rather are the indirect result of reorganization triggerings than the direct product of a reaction to an input. For testing an SO system and, in particular, solving the resulting optimization problem, the limited influence of the tester on the system configuration hence raises implications on the test goal and the test strategy:

1. *Test goal*: The expectation of detected errors for a set of test inputs is strongly associated with the (expected) system reaction on them. The current system state shall consequently have the same influence on $\mathit{\Gamma}$ as on the reaction. Given the internal state space S_{sys} we get the new signature $\mathit{\Gamma}_{\text{SO}} : 2^{I \times S_{\text{sys}}} \to \mathbb{R}$.
2. *Strategy*: An input $i \in I$ at time step t influences future system states by possibly triggering previously unforeseen reconfigurations. Test execution at t thus influences the score, as we call the evaluation of $\mathit{\Gamma}$ for particular inputs, of the following inputs. Test design evolves from an ad hoc towards a sequential decision problem: optimization needs to take into account dynamic interactions between the tester and the system under test (SuT). In consequence, our test goal $\mathit{\Gamma}_{\text{SO}}$ needs to consider input sequences $\pi : T \to I$ over time steps $T = [1..|T|]$ starting from an initial system state $\sigma_0 \in S_{\text{sys}}$ instead of sets of pairs of inputs and system states: $\mathit{\Gamma}_{\text{SO}} : 2^{I^T} \to \mathbb{R}$.

Taken from original publication: André Reichstaller, Thomas Gabor, and Alexander Knapp. Mutation-based test suite evolution for self-organizing systems. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2018



Algorithm 1 Mutation-based test suite evaluation

```
Require: $p \equiv$ reference version of the program under test
         $O \equiv$ set of mutation operators
         $S \equiv$ mutation score function
function killed($\pi$)                          ▷ $\pi$: test case input sequence of length $|\pi|$
    $K_\pi \leftarrow \emptyset$                ▷ map of killed mutants, indexed by mutants
    for all $o \in O$ do
        $m \leftarrow$ mutated version of $p$ by application of $o$
        reset system state
        for $t \leftarrow 1..|\pi|$ do          ▷ iterate through time steps
            $eff \leftarrow p.execute(\pi(t))$
            $eff_m \leftarrow m.execute(\pi(t))$
            if $eff \neq eff_m$ then
                $K_\pi[m] = (eff, eff_m)$       ▷ mutant killed
                break                           ▷ continue with next mutation operator
    return $K_\pi$                              ▷ return killed mutants
function $\Gamma_M(TS)$                         ▷ $TS$: test suite of test input sequences
    $K \leftarrow \emptyset$                    ▷ map of killed mutants, indexed by test input sequences
    for $\pi \in TS$ do                         ▷ iterate through test input sequences
        $K[\pi] \leftarrow$ killed($\pi$)
    return $S(K)$                               ▷ return mutation score
```

For the first implication we previously investigated a mutation-based test goal for SO systems [23]. Following the classic mutation testing technique [9], it determines how many of the mutants that simulate the effect of communication errors during reorganization a test suite reveals. Here, we consider a slightly modified goal not only taking into account the number, but also the effect of revealed mutants on the CCB, which leads to a *weighted* mutation score (Sect. 2). For solving the resulting optimization problem for the second implication, we study the eligibility of meta-heuristic search approaches, or, more concretely, of custom variants of classical evolutionary algorithms. We present a novel evolutionary mutation as well as a recombination operator that are particularly suitable for solving sequential optimization problems (Sect. 3), but may also be useful for various applications beyond test design. An evaluation of the proposed approaches by means of a concrete case, testing a self-organizing, adaptive production cell, shows promising results (Sect. 4). Encouraged by those results and considering related approaches and challenges (Sect. 5), we are planning several combinations and extensions of the presented approaches in the future (Sect. 6).

2 A Mutation-based Test Goal

The mutation testing approach [9] supplies a direct operationalization for the principal goal of finding faults (as effects of errors) through detecting failures: A reference version of the program under test p and a set of *mutation operators* O that mimic particular, common errors are assumed given. Modified versions of p are generated by applying the operators from O. These so-called *mutants* of p simulate the effect of introduced errors

Taken from original publication: André Reichstaller, Thomas Gabor, and Alexander Knapp. Mutation-based test suite evolution for self-organizing systems. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2018

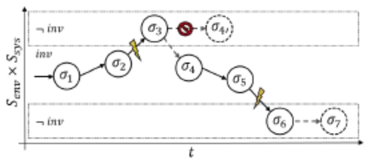


Fig. 2. Corridor of correct behavior

and thus potentially comprise faults. Each of the mutants is executed with each of the test cases from a test suite recording those mutants that show an effect deviating from the effect of p; such a mutant is said to be *killed* by the test case. Finally, for evaluating the test suite, a *mutation score function* S is applied to the record of killed mutants.

Algorithm 1 outlines this mutation-based test suite evaluation $\mathit{\Gamma}_{\mathrm{M}}(TS)$ for a test suite TS of sequences of test cases from the inputs I each of length at most $|T|$. In particular, it considers the case that the execution of one test case in a sequence influences the outcome of the next. We denote the execution of program q with a test case $tc \in I$ on the current system state of S_{sys} by $q.execute(tc)$ returning the observable effect of the resulting state. $\mathit{\Gamma}_{\mathrm{M}}(TS)$ first records the mutation results for each $\pi \in TS$ and then applies the score function S to the computed map of killed mutants. For a test case sequence π, $\mathrm{killed}(\pi)$ executes the program p and each mutant resulting from O from a freshly reset system state with π comparing the effects until either the sequence is completed or the mutant has been detected.

While a classical mutation score function would simply return the overall number of killed mutants, we suggest a more fine-grained categorization of observed effect deviations leading to a *weighted* mutation score. This categorization builds on a basic SO architectural concept: the *corridor of correct behavior* (CCB), for which the remainder of this section suggests appropriate assignments of O and S for testing SO systems.

2.1 The Corridor of Correct Behavior

Our mutation-based testing approach for SO systems assumes that the behavioral specification of the system under test is, or can be, formalized by the *Restore Invariant Approach* (RIA) [17]. Initially introduced as a generic formalism for the specification and the implementation of Organic Computing systems, the RIA and the underlying CCB also proved useful in enabling systematic and automatic tests for SO systems [11]. The CCB guides reorganization in the RIA with logical predicates describing wanted properties of internal and external states. The conjunction of those predicates yields an invariant for the system's specified run-time behavior. Instead of continuously trying to optimize a quantifiable goal, reorganization is only performed if the invariant is broken, i.e., the CCB is left. The reorganization mechanism is assumed to restore the invariant through reorganization then, such that the system can continue to work as expected.

The invariant considers both, the environmental state space S_{env} as well as the internal state space S_{sys}. As soon as changing environmental conditions violate the

Taken from original publication: André Reichstaller, Thomas Gabor, and Alexander Knapp. Mutation-based test suite evolution for self-organizing systems. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2018

invariant, reorganization is triggered by a monitoring mechanism. There are two cases in which the invariant remains broken after reorganization: There might be no possibility for "healing", i.e., re-establishing the invariant by reorganization at all. Since we can say that this is caused by higher force (the system cannot directly control S_{env}), this case is not as relevant for testing. The other case, however, is relevant as it indicates the existence of errors: if there was a possibility for re-establishing the invariant, but the reorganization mechanism did not do that. By testing we strive to reveal this latter case.

The CCB allows us to concentrate the test effort on the reorganization mechanism, as this ensures that the system behavior complies with its specification at run time: A test case tc now amounts to choosing an environmental state, i.e., S_{env} becomes the test input space I. Testing the program p of the reorganization mechanism by executing tc in the current system state, i.e., implementing $p.execute(tc)$, then involves three steps:

1. Establish the particular environmental state tc which, in combination with the current system state violates the invariant.
2. Observe the system's reaction and its effect on the overall state.
3. Evaluate and, where appropriate, classify the effect against the corridor.

2.2 Mutation Operators

Successful reorganization requires correct state perception by the agents, correct computation of internal state adaptations by the reorganization mechanism, and correct realization of the delivered adaptation tasks by the agents. Since all of these critical routines are interconnected through message passing between the agents and the reorganization mechanism, a common cause for system failures are errors in communication. In [23] we elaborated the following exemplary mutation operators which are able to mimic those typical reorganization errors:

– *Lost Reconfiguration Message (LRM)*: As soon as an agent finds that a predicate of the invariant is violated it should normally send a *reconfiguration message* to trigger the reconfiguration mechanism. This mutation operator suppresses such messages, such that there might be no reconfiguration in spite of an incorrect system configuration.
– *Needless Reconfiguration Message (NRM)*: The inverse of an LRM: A particular agent signals the violation of a predicate, although there actually is none. Consequently, unnecessary reconfiguration steps might be triggered.
– *False Reconfiguration (FR)*: This operator mimics the loss of a message that was sent by the reconfiguration mechanism to trigger adaption. In consequence, one agent will (maybe erroneously) retain its previous internal state – this could result again in an incorrect system configuration.

For SO systems, we thus choose the set O of mutation operators to consist of *LRM*, *NRM*, *FR* instantiated to each agent.

2.3 A Weighted Mutation Score

Considering Alg. 1, the final score for TS is determined by a function S which assesses deviations in the observed effects when testing p and when testing the generated mutants.

In its classic form, the score simply counts the number of mutants killed by TS:

$$S_{\mathrm{c}}(K) = |M(K)| \text{ with } M(K) = \{m \mid \exists \pi \in TS \,.\, K[\pi][m] \neq \emptyset\} \,. \tag{1}$$

Besides this classic mutation score we suggest an extension for testing with the CCB, which additionally takes the severity of killed mutants into account. This might be seen as rewarding the test suite for revealing preferably serious failures. The idea behind such a risk markup is that the more serious failures revealed by systematic testing, the lower the probability that such serious failures occur in real operation (cf. [22]).

Considering the CCB we build the severity levels on a classification of possible test results (as we call the effect observed after executing a program with a test case): If test execution results in reorganization, i.e., the reorganization mechanism transferred a state outside the corridor to the inside again, we assign the result to the class $reorg$. Otherwise, we assign it to $\neg reorg$. Function $C : \mathit{Eff} \to \{reorg, \neg reorg\}$ determines the class of an effect in Eff. Comparing the effects eff and eff_m as they are gained in ll. 7 and 8 of Alg. 1, we quantify the severity levels of the four possible permutations with a severity function $Sev : \mathit{Eff} \times \mathit{Eff} \to \mathbb{R}$:

$$Sev(\mathit{eff}, \mathit{eff}_m) = \begin{cases} 1 & \text{if } C(\mathit{eff}) = C(\mathit{eff}_m) \\ 2 & \text{if } C(\mathit{eff}) = reorg \wedge C(\mathit{eff}_m) = \neg reorg \\ 3 & \text{if } C(\mathit{eff}) = \neg reorg \wedge C(\mathit{eff}_m) = reorg \end{cases} \tag{2}$$

The first case is obviously the most harmless one. If a test case triggers reorganization in both program versions (or in neither), we can argue that no real failure was detected. However, as the mutant has been killed (cf. l. 10 in Alg. 1), we still assign a slight severity score. The remaining cases indicate that the killed mutant simulated a real failure. In the second case, no valid state was established even though this would be possible. Such a failure would require human intervention in real operation. The third case, where a valid state was established even though this should be impossible, implies even higher costs in real operation, as it mostly results in a contradiction between software and hardware. The quantified severity levels are the basis for the weighted mutation score

$$S_{\mathrm{w}}(K) = \textstyle\sum_{m \in M(K)} Agg(\{\!\{ Sev(K[\pi][m]) \mid \pi \in TS \}\!\}) \,, \tag{3}$$

where Agg aggregates the multiset of severities observed when a single mutant was killed by more than one test case occurring TS. The operator Agg can be instantiated, e.g., with $\sum$ or $\max$. We suggest to use $\sum$ if the errors simulated by the mutation operator are assumed to be *transient*, which means that the error does not always trigger a failure if covered. For the others, the *persistent* errors, we suggest to use $\max$.

3 Evolutionary Test Strategies

Given a mutation-based test goal Γ_{M} as it is implemented in Alg. 1 and instantiated with mutation operators O and mutation score function S, just as described in the previous section, the challenge is now to find a test suite TS that optimizes this goal in terms of the obtained score. We further demand that $|TS|$ conforms with a predefined maximum

Taken from original publication: André Reichstaller, Thomas Gabor, and Alexander Knapp. Mutation-based test suite evolution for self-organizing systems. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2018

$$\begin{tikzcd}
 & & \text{initial} \arrow[dl, "tc_1"'] \arrow[dr, "tc_2"] & & \\
 & \mathit{eff}_1\{m_0[2], m_2[1]\} & & \mathit{eff}_2\{m_4[2]\} \arrow[dl, "tc_3"'] \arrow[dr, "tc_7"] & \\
 & & \mathit{eff}_3\{m_2[3], m_5[1]\} \arrow[dll, "tc_4"'] \arrow[d, "tc_5"] \arrow[drr, "tc_6"] & & \mathit{eff}_7\{m_2[1], m_6[2], m_7[3]\} \\
\mathit{eff}_4\{m_1[1]\} & & \mathit{eff}_5\{m_3[1], m_6[1], m_8[1]\} & & \mathit{eff}_6\{m_9[3], m_7[1]\}
\end{tikzcd}$$

Fig. 3. Exemplary dependency graph with seven test cases $tc_1, \ldots, tc_7$ annotated by the set of killed mutants respectively. The numbers in squared brackets denote the severity of killed mutants. The best test suite of size $|TS| = 2$ for $\max$-aggregation comprises of the two sequences $\pi_4 = \langle tc_2, tc_3, tc_6 \rangle$ and $\pi_5 = \langle tc_2, tc_7 \rangle$ scoring 14. For $\sum$-aggregation, however, the best test suite with $|TS| = 2$ is $\{\pi_3, \pi_5\}$ with $\pi_3 = \langle tc_2, tc_3, tc_5 \rangle$ with a score of 19.

number of investable time steps k such that if each test sequence of TS has a length of (at most) $|T|$, $k = |TS| \cdot |T|$. In case of a self-organizing SuT the search for such a test suite has to face the following two challenges:

1. The effects of test cases in terms of killed mutants are dependent on the full history of previously executed test cases in a test sequence due to reconfigurations as adaptations to these previous environmental influences; in particular, executing a test case influences the future scores. The search space for the optimal test suite is thus given by a *dependency graph* with the initial system state as root, effects and their killed mutants as nodes, and the test cases as edges; see Fig. 3 for a small example.
2. Each evaluation of a test suite TS is at the cost of $k \cdot |O|$ program executions at worst. The only factor that we can influence for practicability is thus the number of evaluations that has to be kept to a minimum.

These challenges give rise to a general optimization problem: find a number of paths through a graph in the most efficient way, such that their collected nodes optimize a given goal. For the aggregation by $\sum$, when disregarding that test suites are sets, a single best path could just be repeated, and optimization would be reduced to the well-established problem of finding a single path with maximum score [20]. The aggregation operator $\max$, however, directly considers sets of nodes for evaluation and is sensitive to duplicates; greedy approaches iteratively choosing the single best rated path are doomed to fail. We now report on some experiments with different evolutionary algorithms for mastering this problem. Utilizing the new technique of *phased evolution* (cf. Sect. 3.2) we manage to cut the number of needed goal evaluations; endowing the evolutionary mutation and recombination operators with domain specific semantics derived from a similarity function between test cases, we leverage the classical evolutionary algorithm to cope with the data structure of test sequences in *penguin evolution* (cf. Sect. 3.3).

Taken from original publication: André Reichstaller, Thomas Gabor, and Alexander Knapp. Mutation-based test suite evolution for self-organizing systems. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2018

3.1 Evolutionary Algorithms

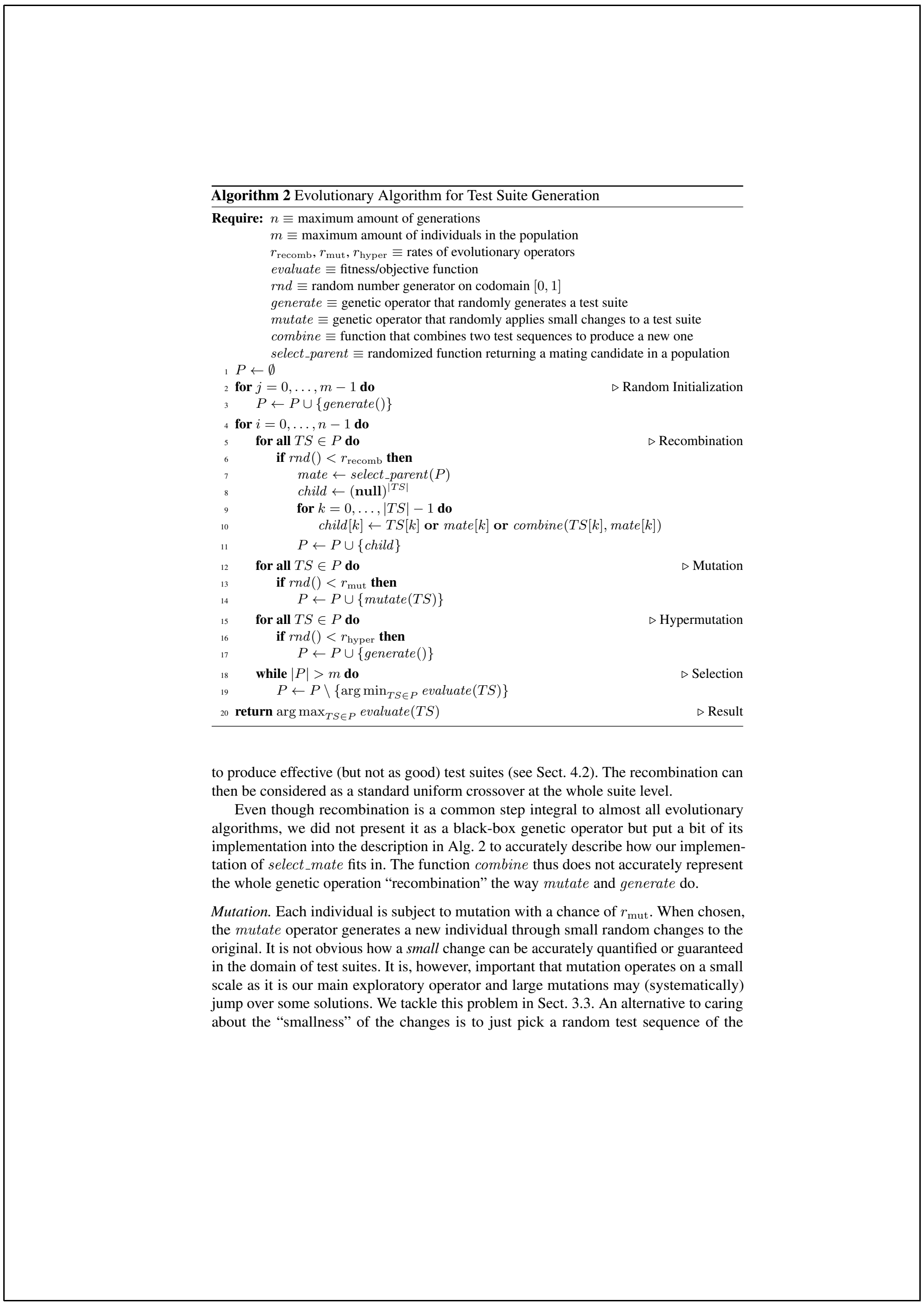
Evolutionary algorithms are a wide-spread probabilistic optimization technique [12]. As they do not require a gradient on the solution space to be computable, they are often used in the automatic generation of test cases for software [20, 26]. In more recent years, the research community considered the issue of *whole test suite generation*, in which the aim of applying an evolutionary algorithm is not to find the most important test cases but instead to find the ideal combination of test cases that make up a concise but approximately complete test suite for a given software [13].

We first discuss the basis of an evolutionary process for whole test suite generation, which we will augment in the following sections. Any evolutionary algorithm works on a set of solution candidates, also called *individuals*. In our case, a single individual TS represents a whole test suite. The set of currently considered individuals is also called a *population* P and thus forms a subset of the domain of all possible test suites. As is usual for evolutionary algorithms, we set a fixed limit $|P| = m$ on the population size. Furthermore, we employ a fixed limit of execution time (measured in evaluations or generations as we discuss later) instead of a quality threshold as would be possible as well. However, especially for our later experiments we are most interested in the comparison of the quality of various approaches within a given time frame, as for software testing the requirement is more likely formulated to produce the best test suite within the available time rather than to produce a test suite as fast as possible. Algorithm 2 shows the typical structure of such an evolutionary algorithm. It starts with a random initialization and repeats its other operations for a fixed amount of times n. Each of these repetitions is also called a *generation*. We will discuss the various operators in greater detail now.

Random Initialization. This step generates the initial population by generating random test suites. Note that $generate$ is not a mathematical function as it returns a newly generated object each time it is called. We use the term *genetic operator* for common evolutionary operations that use random effects.

Recombination. We chose a variant of recombination that grants the chance to recombine to each individual (irregardless of its fitness), but chooses its respective mate with respect to higher fitness. Effectively, we found this to be a good compromise between allowing exploration (using all individuals for recombination) and exploitation (favoring the better ones). The former is guaranteed by applying a fixed chance r_{recomb} for the choice of any individual for recombination. The randomized function $select\_mate$ performs the latter by iterating over the population, returning the nth-fittest individual with probability 2^{-n}. We then first create an empty test suite (i.e., containing no test sequences but already of required size $|TS|$) in the variable $child$. We then iterate over the number of test suites that is used for all our suites and complete the child by performing one random choice of three operations with equal probability (as denoted by the $\mathbf{or}$ operator): (a) we reuse the test sequence of the first (randomly chosen) parent, or (b) we use the test sequence of the second (chosen according to fitness) parent, or (c) we call a special function $combine$ that builds a new test sequence out of the test sequences stemming from both parents. We show in Sect. 3.3 how to effectively implement such a function. Leaving out option (c) entirely would result in a more standard evolutionary algorithm that still manages

Taken from original publication: André Reichstaller, Thomas Gabor, and Alexander Knapp. Mutation-based test suite evolution for self-organizing systems. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2018



Algorithm 2 Evolutionary Algorithm for Test Suite Generation

```
Require: $n \equiv$ maximum amount of generations
         $m \equiv$ maximum amount of individuals in the population
         $r_{\text{recomb}}, r_{\text{mut}}, r_{\text{hyper}} \equiv$ rates of evolutionary operators
         $evaluate \equiv$ fitness/objective function
         $rnd \equiv$ random number generator on codomain $[0, 1]$
         $generate \equiv$ genetic operator that randomly generates a test suite
         $mutate \equiv$ genetic operator that randomly applies small changes to a test suite
         $combine \equiv$ function that combines two test sequences to produce a new one
         $select\_parent \equiv$ randomized function returning a mating candidate in a population
$P \leftarrow \emptyset$
for $j = 0, \ldots, m - 1$ do                                   ▷ Random Initialization
    $P \leftarrow P \cup \{generate()\}$
for $i = 0, \ldots, n - 1$ do
    for all $TS \in P$ do                                        ▷ Recombination
        if $rnd() < r_{\text{recomb}}$ then
            $mate \leftarrow select\_parent(P)$
            $child \leftarrow (\textbf{null})^{|TS|}$
            for $k = 0, \ldots, |TS| - 1$ do
                $child[k] \leftarrow TS[k]$ or $mate[k]$ or $combine(TS[k], mate[k])$
            $P \leftarrow P \cup \{child\}$
    for all $TS \in P$ do                                        ▷ Mutation
        if $rnd() < r_{\text{mut}}$ then
            $P \leftarrow P \cup \{mutate(TS)\}$
    for all $TS \in P$ do                                        ▷ Hypermutation
        if $rnd() < r_{\text{hyper}}$ then
            $P \leftarrow P \cup \{generate()\}$
    while $|P| > m$ do                                           ▷ Selection
        $P \leftarrow P \setminus \{\arg\min_{TS \in P} evaluate(TS)\}$
return $\arg\max_{TS \in P} evaluate(TS)$                         ▷ Result
```

to produce effective (but not as good) test suites (see Sect. 4.2). The recombination can then be considered as a standard uniform crossover at the whole suite level.

Even though recombination is a common step integral to almost all evolutionary algorithms, we did not present it as a black-box genetic operator but put a bit of its implementation into the description in Alg. 2 to accurately describe how our implementation of *select_mate* fits in. The function *combine* thus does not accurately represent the whole genetic operation "recombination" the way *mutate* and *generate* do.

Mutation. Each individual is subject to mutation with a chance of r_{mut}. When chosen, the *mutate* operator generates a new individual through small random changes to the original. It is not obvious how a *small* change can be accurately quantified or guaranteed in the domain of test suites. It is, however, important that mutation operates on a small scale as it is our main exploratory operator and large mutations may (systematically) jump over some solutions. We tackle this problem in Sect. 3.3. An alternative to caring about the "smallness" of the changes is to just pick a random test sequence of the suite and re-generate it through random walk within the dependency graph starting at a randomly chosen point in the test sequence, which results in a rather big change with each mutation. We compare these approaches in Sect. 4.2. Note that in contrast to some evolutionary algorithms (and biological evolution), we only add mutated individuals to the population instead of having them replace their original counterparts.

Hypermutation. During the hypermutation step, we simply generate new individuals at random disregarding the previous course of evolution, and add them to the population. For this purpose, we use the same $generate$ operator as in the random initialization step. Adding these new individuals increases exploratory behavior and thus helps prevent getting stuck in local optima. In parallel to the other operators (and their respective application rates), we base the amount of generated individuals on the population size $|P|$ and the given parameter r_{hyper}. In fact, the phases extension discussed in Sect. 3.2 turns the evolutionary algorithm into a dynamic optimization problem, for which the use of hypermutation has been highly suggested [16].

Selection. In the selection step we simply choose the m best individuals to keep for the next generation. For the description in Alg. 2 we choose a notation that does not need to introduce list slicing, although the implementation uses a computationally more efficient functional equivalent to the algorithm presented here.

Result. Finally, we return the best individual found in the last population of the last generation. This is also the best individual found overall, as all of our operations in the steps within each generation only add new individuals but never overwrite their parents. The fitness function $evaluate$ in our case applies Γ_{M} to the TS. As long as we do not change its semantics this means that we always keep the best individuals around. This feature is called *elitism* within the field of evolutionary algorithms. While the search process is (even without elitism) expected to strive for better individuals anyway, elitism ensures that it is monotone, as we will see in Sect. 4.2.

3.2 Phases Extension

Having discussed the basic functionality of our evolutionary algorithm for test suite generation, we now introduce the first of two extensions to it. This extension considers improving the performance of the search. We show in Sect. 4.2 that it manages to produce comparable results with roughly half the goal evaluations.

Generally speaking, we can observe that there is a noticeable relation between the fitness of a test suite TS and the fitness of a single test sequence $\pi \in TS$, i.e., the fitness of the suite $\{\pi\}$. As discussed, the best test suite of x test sequences (with maximum length $|T|$) will usually not consist of the x best rated test sequences, as these will likely overlap in killed mutants and thus have poor overall coverage. However, it seems intuitive to start with one of the best rated test sequences and then build a suite around it. We could thus split the test suite generation problem into various sub-problems of iteratively finding test sequences given certain constraints (from previously found test sequences). But evolutionary algorithms provide us with a much more elegant approach, which we call *phase-based evolution*: we adjust the objective of the evolutionary process and the data structure of its individuals during the progression of evolutionary search.

Taken from original publication: André Reichstaller, Thomas Gabor, and Alexander Knapp. Mutation-based test suite evolution for self-organizing systems. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2018

We start our evolutionary process with individuals that contain test suites $TS \in P$ of size $|TS| = 1$, i.e., all test suites only contain a single test sequence. We run this evolutionary search for the best single test sequence for a while: if we eventually want to search for a test suite TS of size $|TS| = x$ after n generations, we run this reduced search problem for roughly $\frac{n}{x}$ generations. Then we augment all individuals to represent a test suite with two test sequences by adding a randomly generated test sequence to each individual. We proceed to expand the problem domain of the search every $\frac{n}{x}$ generations until generation n, having actually employed the original fitness function for a size x test suite only for the last $\frac{1}{n}$ generations. This approach works well in case the time of each of these evolutionary phases does run long enough to find reasonable results but not long enough to fully converge. The evolutionary search thus hits a point where it has a rough idea about the best single test sequence but still has multiple open options. At this point, it proceeds to search for a larger test suite, with limited option for the first spot of a test suite. Using the phases extension, we can cut the total amount of goal evaluations roughly in half, since the average test suite is only $\frac{x+1}{2}$ test sequences in size throughout the course of evolution.

3.3 Penguin Extensions

The second extension targets two points: (1) “merging” two test sequences into one within the *combine* function and (2) applying meaningfully small changes in the search domain of test suites within the *mutate* operator. The main problem of both is the handling of test case dependencies: Two test sequences π_1 and π_2 cannot simply be combined by attaching the tail of π_2 to the head of π_1 (as in the traditional one-point crossover operation [12]), since the configurations in the second half of π_2 might not conform to those of the first half of π_1. We utilize a method we call *penguin recombination* instead. Its name is inspired by an imaginary instance of our evolutionary algorithm being used to compute the evolution of animals, where dependencies in combination and mutation can be observed as well. If we consider two test sequences as different species such as a parrot and a fish, we notice that they cannot meaningfully recombine through crossover; but, inspired by nature, we can at least evolve the parrot to another bird that is most similar to the fish, resulting in perhaps a penguin. For applying this metaphor to the test sequences considered, we utilized a notion of similarity which we introduced in [23].

Similarity between Test Sequences. We showed that faults that result from errors which are emulated by the mutation operators mentioned in Sect. 2.2 have no influence on one another, as they are distributed over different entities that are only connected through message passing. This basically means that a fault in the source code of one agent does not affect the path passed through control flow of another. We showed that we are in this case able to determine which of all the possibly generated faults F (as result of applying mutation operators) a test case would generally cover [23]. We can thus represent a test case by a *label vector*, a binary vector v of length $|F|$, with $v_i = 1$ if the fault F_i is covered and $v_i = 0$ otherwise. Such a vector can be viewed as indicator for the path taken through the distributed control flow in an SO system in response to the input of an executed test case. Building on this insight we proposed a dissimilarity metric comparing

Taken from original publication: André Reichstaller, Thomas Gabor, and Alexander Knapp. Mutation-based test suite evolution for self-organizing systems. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2018

two test cases based on their label vectors v_1 and v_2:

$$\mathit{Dist}(v_1, v_2) = |\{l \in \{1, \ldots, n\} \mid v_1[l] \neq v_2[l]\}| \ . \tag{4}$$

The more dissimilar two test cases are w.r.t. (4) the more fruitful it might be to execute them both instead of only one of them. Writing $tc.v$ for the label vector of test case tc, we extended this metric for assessing the representativeness or similarity of whole test sequences π_1 and π_2, i.e., paths through the dependency graph by

$$\mathit{Rep}(\pi_1, \pi_2) = \textstyle\sum_{tc_2 \in \pi_2} \min_{tc_1 \in \pi_1} \mathit{Dist}(tc_1.v, tc_2.v) \ . \tag{5}$$

Penguin Recombination. Using this path distance, two test sequences π_1 and π_2 can be combined as follows: we cut a part of the beginning of π_1 at a random length, resulting in the incomplete test sequence π_1^A so that $\pi_1^A; \pi_1^B = \pi_1$ for some π_1^B. There now exist multiple paths that may follow, of which π_1^B is one possibility. Of all the possibilities $\pi_1^B, \pi_1^{B'}, \ldots$ within the current configuration at π_1^A, we compute their similarity to π_2^B, which is the second part of π_2 after cutting off π_2 at the same length as π_1. We choose the most similar completion $\pi_1^{B^*} \sim \pi_2^B$ to produce a new test sequence $\pi_3 = \pi_1^A; \pi_1^{B^*}$. This test sequence has a similar setup as π_1 but after a certain point tries to mimic as many features of π_2 as possible, i.e., become as much of a fish as a parrot can.

Penguin Mutation. The mutation operator is implemented analogously, almost as a recombination of a test sequence π with itself. We cut off π at some random point, resulting in $\pi^A; \pi^B = \pi$. Furthermore, we cut off the first test case of π^B, resulting in $\pi^B_{\mathrm{orig}}; \pi^C = \pi^B$. We then add one test case at random to π^A, which we name π^B_{rand}, and make sure that $\pi^B_{\mathrm{rand}} \neq \pi^B_{\mathrm{orig}}$. From that point on, we complete $\pi^A; \pi^B_{\mathrm{rand}}$ by generating the test sequence $\pi^{C^*} \sim \pi^C$. We return the mutated test sequence $\pi' = \pi^A; \pi^B_{\mathrm{rand}}; \pi^{C^*}$ with only a single test case changed and afterwards trying to mimic the original π as closely as still possible. We argue that this is the minimal (and still general) mutation one can implement for the domain of test sequences.

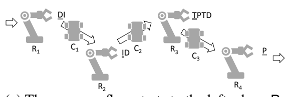
4 Evaluation

We evaluated the presented approaches by means of a concrete case study of a self-organizing, adaptive production cell. After describing the case considered in Sect. 4.1 we will provide the results in Sect. 4.2.

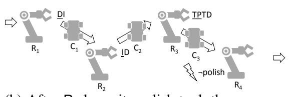
4.1 Case Study: An Adaptive Production Cell

We consider evolving a test suite for a self-organizing, adaptive production cell. As depicted in Fig. 4, the considered setup comprises four robots (R1, R2, R3, R4) and three mobile carts (C1, C2, C3). The robots are equipped with tools and corresponding capabilities such as Drill, Insert, Tighten, and Polish. The carts are able to carry workpieces along given routes. Each of the robots can be associated with a particular role which lets it apply a sequence of capabilities on present workpieces. The self-organizing production cell's behavior at a point in time t is thus determined by the overall role allocation and

Taken from original publication: André Reichstaller, Thomas Gabor, and Alexander Knapp. Mutation-based test suite evolution for self-organizing systems. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2018



(a) The resource flow starts to the left where R_1 drills a hole into incoming workpieces. Workpieces are successively transported by C_1, C_2, and C_3. The robots apply their tools. Once R_4 is done, the workpieces leave the system.



(b) After R_4 loses its polish tool, the resource flow is reconfigured: R_3 is taking over the previous role of R_4.

Fig. 4. A schematic overview of the self-organizing robot cell case study. The task is to apply the drill, insert, tighten, and polish capabilities to all incoming workpieces. Each robot's available tools are shown to its right with D, I, T, and P; the currently allocated ones are underlined. Figure 4a shows an exemplary configuration of the robot cell. As depicted in Fig. 4b, faults result in tool losses that self-organization can cope with by reconfiguring the resource flow.

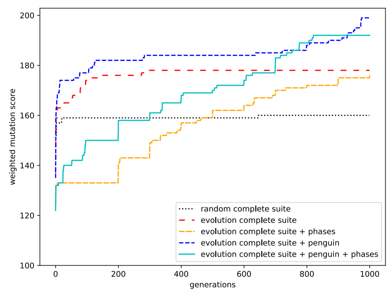
cart routes in t. A corridor of correct behavior (cf. Sect. 2.1) monitors the satisfiability of tasks in the presence of environmental faults, such as a broken driller for a specific robot. Triggered by violations an SO mechanism calculates and distributes a new valid configuration at run-time.

Testing the SO mechanism in the described setup means to simulate environmental faults by use of test drivers in order to subsequently evaluate the established, new configuration of the cell. While the inputs of a single test case are defined by a fixed number of environmental faults (55 in our case), a test sequence starting at a fixed role allocation with no activated faults, sequentially activates the faults defined by the comprised test cases. The result of a test case depends on the current role allocation, which can be viewed as an internal system state, and this current role allocation is established by the preceding test cases. The dependency graph of test cases hence is connected by role allocations before and after the test case execution. For our experiments, we generated the test suite to minimize by use of the S# framework [10] resulting in a graph with 7524 test cases as nodes and 8 884 634 edges in between them.

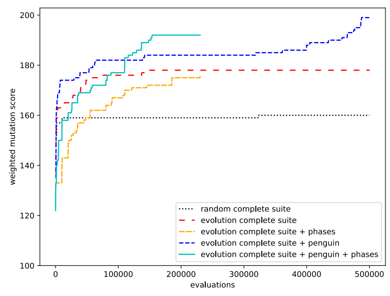
4.2 Results

For evaluation we applied our approach to the mentioned dependency graph. We tested a standard evolutionary algorithm evolving a test suite as well as both of our extensions individually and their combination. We also ran baseline experiments using random search. We used a population of size $m = 50$ evolving for $n = 1000$ generations. We produced test suites of (eventual) size $x = 10$ from our test data comprising test sequences of length up to $|T| = 10$, i.e., $k = 100$. We chose $r_{\text{recomb}} = 0.3$ and $r_{\text{mut}} = r_{\text{hyper}} = 0.1$ for the hyperparameters providing a lot of random exploration to the algorithm favoring generality of our results over sample efficiency. The total computation time of all evolutionary processes included in the test was $1.6\,\text{h}$ on a machine with an Intel Core i7 processor at $2.9\,\text{GHz}$ and $16\,\text{GB}$ of memory. The results are shown in Figs. 5a and 5b. It can be clearly seen that the phases extension eventually

Taken from original publication: André Reichstaller, Thomas Gabor, and Alexander Knapp. Mutation-based test suite evolution for self-organizing systems. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2018



(a) Performance evaluation of evolutionary test generations strategies compared to a random baseline. The penguin recombination and mutation manages to produce better results than the naïve approach. Interestingly, for both cases, about the same quality of results can be reached using the phased evolutionary algorithm with significantly less computational effort, see Fig. 5b.



(b) The same experiment as in Fig. 5a plotted against the number of test sequence evaluations performed. It can be seen that the phase-based extension uses only about half the evaluations compared to the standard approach, reaching about the same performance earlier in the case of the penguin variant.

Fig. 5. Performance evaluation results

Taken from original publication: André Reichstaller, Thomas Gabor, and Alexander Knapp. Mutation-based test suite evolution for self-organizing systems. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2018

achieves very similar results to both non-phase-based variants, but with considerable savings in computational resources. Furthermore, it is also evident that both penguin variants outperform their non-penguin counterparts. Again, this validates our approach and shows that the additional knowledge given to the algorithm in form of the similarity function pays off with better end results.

5 Related Work

Extensive use of adaptivity, such as self-organization, necessitates research on adequate methods of engineering them reliably [8, 28]. Finding adequately powerful software tests is an integral part of making adaptive systems controllable and trustworthy [1, 3, 6], even though some new methods regarding, e.g., run-time testing [4, 7, 14], need to be developed. The concept of simultaneously using machine learning methods to generate the adaptivity of the SuT as well as the power of the test suite has been sketched for neural networks as *adversarial learning* [2, 19].

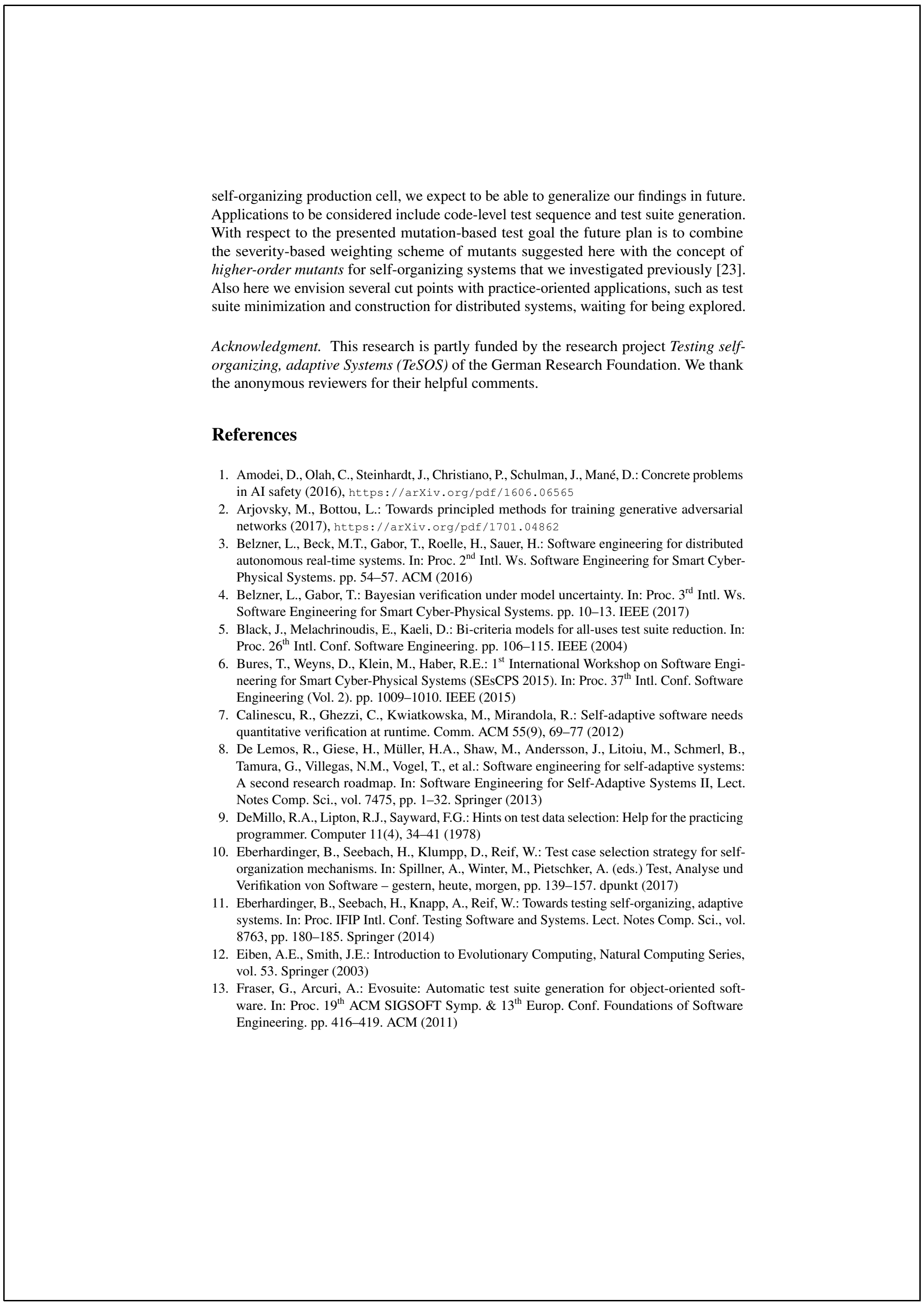
We considered test suites for self-organizing systems, and the application of evolutionary algorithms for their generation w.r.t. a mutation-based test goal. While it seems quite common to use fault-based techniques for evaluating the quality of test suites [5, 24, 29], the huge majority of approaches, including the cited ones, applies other test goals for actual generation or the minimization process. This might be due to the high costs for goal evaluation, which we were able to reduce by the phases extension. The mutation operators and the case study were taken from our previous work, where we considered the test suite reduction problem for SO systems [23]. Also the severity-based mutant weighting was inspired by our previous work in which we approached the task of risk-based interoperability testing using reinforcement learning [22].

Here, we made use of search-based testing techniques [20] for generating test suites which are adequate w.r.t. a mutation-based test goal. Within the field of evolutionary algorithms, test case generation has been researched for some time [20, 26] with whole suite generation sparking interest more recently [13]. Using evolutionary algorithms for dynamically changing problems has been envisioned from their very beginning [12, 27]. Some approaches have already introduced dynamics into originally non-dynamic problems in order to improve the quality of the search result [15, 25]. These also use measurements related to the similarity between individuals in their evaluation, which may then change over time as the population changes. Similarity has been incorporated into the recombination process e.g. in [18], though on a different level than in our approach, viz. at the level of mate selection mirroring biological evolution.

6 Conclusion

We suggested two domain specific extensions of a classical evolutionary approach on constructing test suites of given length w.r.t. a mutation-based test goal for testing self-organizing systems. The first, the *phased* extension, reduced the number of goal evaluations needed for optimization, the second, the *penguin* extension, was shown to increase the overall fitness attained. Both aspects are highly relevant for test suite construction. Though our evaluation just considered a single concrete case, testing a

Taken from original publication: André Reichstaller, Thomas Gabor, and Alexander Knapp. Mutation-based test suite evolution for self-organizing systems. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2018



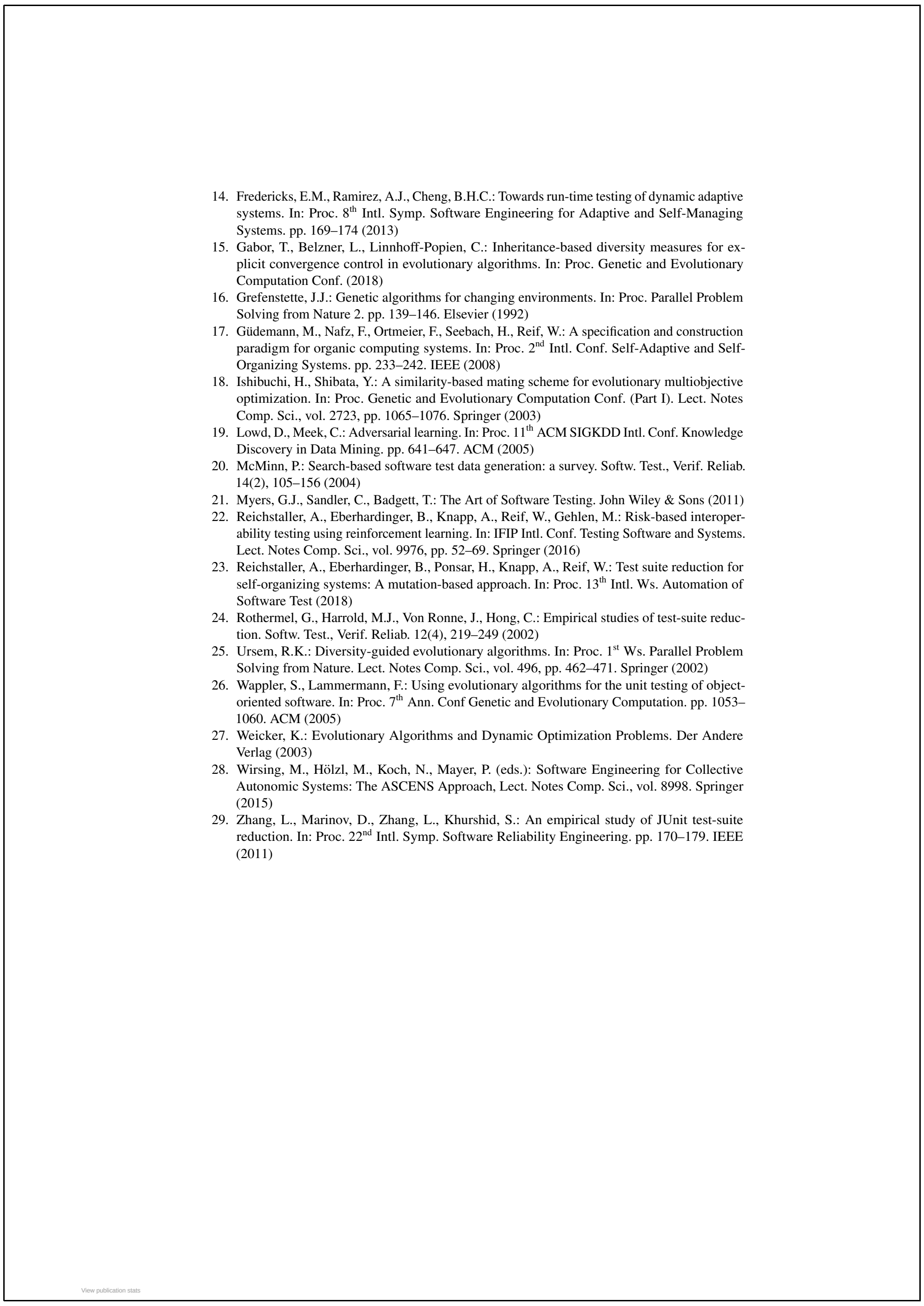
self-organizing production cell, we expect to be able to generalize our findings in future. Applications to be considered include code-level test sequence and test suite generation. With respect to the presented mutation-based test goal the future plan is to combine the severity-based weighting scheme of mutants suggested here with the concept of *higher-order mutants* for self-organizing systems that we investigated previously [23]. Also here we envision several cut points with practice-oriented applications, such as test suite minimization and construction for distributed systems, waiting for being explored.

Acknowledgment. This research is partly funded by the research project *Testing self-organizing, adaptive Systems (TeSOS)* of the German Research Foundation. We thank the anonymous reviewers for their helpful comments.

References

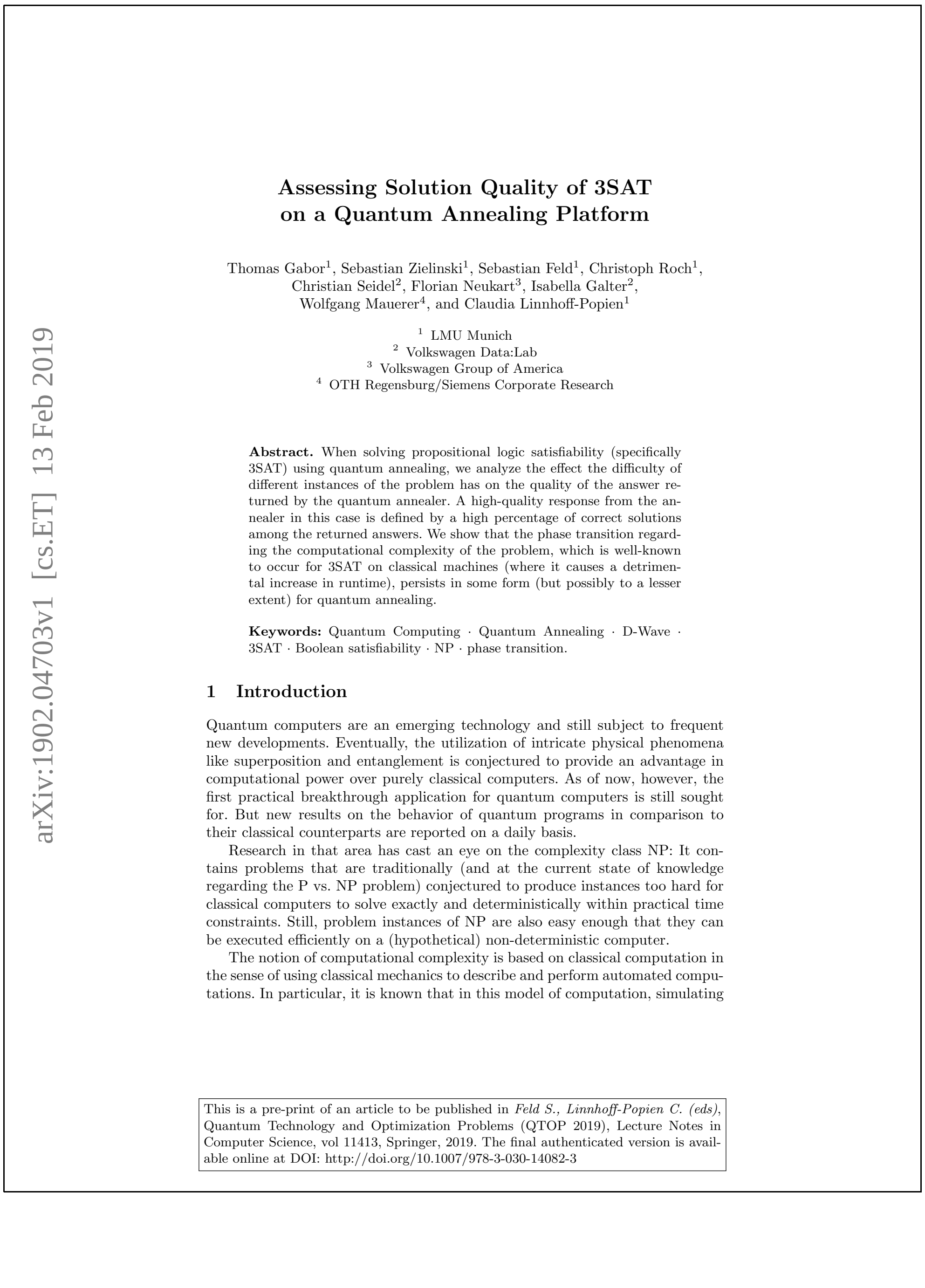
1. Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., Mané, D.: Concrete problems in AI safety (2016), https://arXiv.org/pdf/1606.06565
2. Arjovsky, M., Bottou, L.: Towards principled methods for training generative adversarial networks (2017), https://arXiv.org/pdf/1701.04862
3. Belzner, L., Beck, M.T., Gabor, T., Roelle, H., Sauer, H.: Software engineering for distributed autonomous real-time systems. In: Proc. 2nd Intl. Ws. Software Engineering for Smart Cyber-Physical Systems. pp. 54–57. ACM (2016)
4. Belzner, L., Gabor, T.: Bayesian verification under model uncertainty. In: Proc. 3rd Intl. Ws. Software Engineering for Smart Cyber-Physical Systems. pp. 10–13. IEEE (2017)
5. Black, J., Melachrinoudis, E., Kaeli, D.: Bi-criteria models for all-uses test suite reduction. In: Proc. 26th Intl. Conf. Software Engineering. pp. 106–115. IEEE (2004)
6. Bures, T., Weyns, D., Klein, M., Haber, R.E.: 1st International Workshop on Software Engineering for Smart Cyber-Physical Systems (SEsCPS 2015). In: Proc. 37th Intl. Conf. Software Engineering (Vol. 2). pp. 1009–1010. IEEE (2015)
7. Calinescu, R., Ghezzi, C., Kwiatkowska, M., Mirandola, R.: Self-adaptive software needs quantitative verification at runtime. Comm. ACM 55(9), 69–77 (2012)
8. De Lemos, R., Giese, H., Müller, H.A., Shaw, M., Andersson, J., Litoiu, M., Schmerl, B., Tamura, G., Villegas, N.M., Vogel, T., et al.: Software engineering for self-adaptive systems: A second research roadmap. In: Software Engineering for Self-Adaptive Systems II, Lect. Notes Comp. Sci., vol. 7475, pp. 1–32. Springer (2013)
9. DeMillo, R.A., Lipton, R.J., Sayward, F.G.: Hints on test data selection: Help for the practicing programmer. Computer 11(4), 34–41 (1978)
10. Eberhardinger, B., Seebach, H., Klumpp, D., Reif, W.: Test case selection strategy for self-organization mechanisms. In: Spillner, A., Winter, M., Pietschker, A. (eds.) Test, Analyse und Verifikation von Software – gestern, heute, morgen, pp. 139–157. dpunkt (2017)
11. Eberhardinger, B., Seebach, H., Knapp, A., Reif, W.: Towards testing self-organizing, adaptive systems. In: Proc. IFIP Intl. Conf. Testing Software and Systems. Lect. Notes Comp. Sci., vol. 8763, pp. 180–185. Springer (2014)
12. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing, Natural Computing Series, vol. 53. Springer (2003)
13. Fraser, G., Arcuri, A.: Evosuite: Automatic test suite generation for object-oriented software. In: Proc. 19th ACM SIGSOFT Symp. & 13th Europ. Conf. Foundations of Software Engineering. pp. 416–419. ACM (2011)

Taken from original publication: André Reichstaller, Thomas Gabor, and Alexander Knapp. Mutation-based test suite evolution for self-organizing systems. In *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2018



14. Fredericks, E.M., Ramirez, A.J., Cheng, B.H.C.: Towards run-time testing of dynamic adaptive systems. In: Proc. 8th Intl. Symp. Software Engineering for Adaptive and Self-Managing Systems. pp. 169–174 (2013)
15. Gabor, T., Belzner, L., Linnhoff-Popien, C.: Inheritance-based diversity measures for explicit convergence control in evolutionary algorithms. In: Proc. Genetic and Evolutionary Computation Conf. (2018)
16. Grefenstette, J.J.: Genetic algorithms for changing environments. In: Proc. Parallel Problem Solving from Nature 2. pp. 139–146. Elsevier (1992)
17. Güdemann, M., Nafz, F., Ortmeier, F., Seebach, H., Reif, W.: A specification and construction paradigm for organic computing systems. In: Proc. 2nd Intl. Conf. Self-Adaptive and Self-Organizing Systems. pp. 233–242. IEEE (2008)
18. Ishibuchi, H., Shibata, Y.: A similarity-based mating scheme for evolutionary multiobjective optimization. In: Proc. Genetic and Evolutionary Computation Conf. (Part I). Lect. Notes Comp. Sci., vol. 2723, pp. 1065–1076. Springer (2003)
19. Lowd, D., Meek, C.: Adversarial learning. In: Proc. 11th ACM SIGKDD Intl. Conf. Knowledge Discovery in Data Mining. pp. 641–647. ACM (2005)
20. McMinn, P.: Search-based software test data generation: a survey. Softw. Test., Verif. Reliab. 14(2), 105–156 (2004)
21. Myers, G.J., Sandler, C., Badgett, T.: The Art of Software Testing. John Wiley & Sons (2011)
22. Reichstaller, A., Eberhardinger, B., Knapp, A., Reif, W., Gehlen, M.: Risk-based interoperability testing using reinforcement learning. In: IFIP Intl. Conf. Testing Software and Systems. Lect. Notes Comp. Sci., vol. 9976, pp. 52–69. Springer (2016)
23. Reichstaller, A., Eberhardinger, B., Ponsar, H., Knapp, A., Reif, W.: Test suite reduction for self-organizing systems: A mutation-based approach. In: Proc. 13th Intl. Ws. Automation of Software Test (2018)
24. Rothermel, G., Harrold, M.J., Von Ronne, J., Hong, C.: Empirical studies of test-suite reduction. Softw. Test., Verif. Reliab. 12(4), 219–249 (2002)
25. Ursem, R.K.: Diversity-guided evolutionary algorithms. In: Proc. 1st Ws. Parallel Problem Solving from Nature. Lect. Notes Comp. Sci., vol. 496, pp. 462–471. Springer (2002)
26. Wappler, S., Lammermann, F.: Using evolutionary algorithms for the unit testing of object-oriented software. In: Proc. 7th Ann. Conf Genetic and Evolutionary Computation. pp. 1053–1060. ACM (2005)
27. Weicker, K.: Evolutionary Algorithms and Dynamic Optimization Problems. Der Andere Verlag (2003)
28. Wirsing, M., Hölzl, M., Koch, N., Mayer, P. (eds.): Software Engineering for Collective Autonomic Systems: The ASCENS Approach, Lect. Notes Comp. Sci., vol. 8998. Springer (2015)
29. Zhang, L., Marinov, D., Zhang, L., Khurshid, S.: An empirical study of JUnit test-suite reduction. In: Proc. 22nd Intl. Symp. Software Reliability Engineering. pp. 170–179. IEEE (2011)

Taken from original publication: Thomas Gabor, Sebastian Zielinski, Sebastian Feld, Christoph Roch, Christian Seidel, Florian Neukart, Isabella Galter, Wolfgang Mauerer, and Claudia Linnhoff-Popien. Assessing solution quality of 3SAT on a quantum annealing platform. In *International Workshop on Quantum Technology and Optimization Problems*. Springer, 2019



Assessing Solution Quality of 3SAT on a Quantum Annealing Platform

Thomas Gabor[1], Sebastian Zielinski[1], Sebastian Feld[1], Christoph Roch[1], Christian Seidel[2], Florian Neukart[3], Isabella Galter[2], Wolfgang Mauerer[4], and Claudia Linnhoff-Popien[1]

[1] LMU Munich
[2] Volkswagen Data:Lab
[3] Volkswagen Group of America
[4] OTH Regensburg/Siemens Corporate Research

Abstract. When solving propositional logic satisfiability (specifically 3SAT) using quantum annealing, we analyze the effect the difficulty of different instances of the problem has on the quality of the answer returned by the quantum annealer. A high-quality response from the annealer in this case is defined by a high percentage of correct solutions among the returned answers. We show that the phase transition regarding the computational complexity of the problem, which is well-known to occur for 3SAT on classical machines (where it causes a detrimental increase in runtime), persists in some form (but possibly to a lesser extent) for quantum annealing.

Keywords: Quantum Computing · Quantum Annealing · D-Wave · 3SAT · Boolean satisfiability · NP · phase transition.

1 Introduction

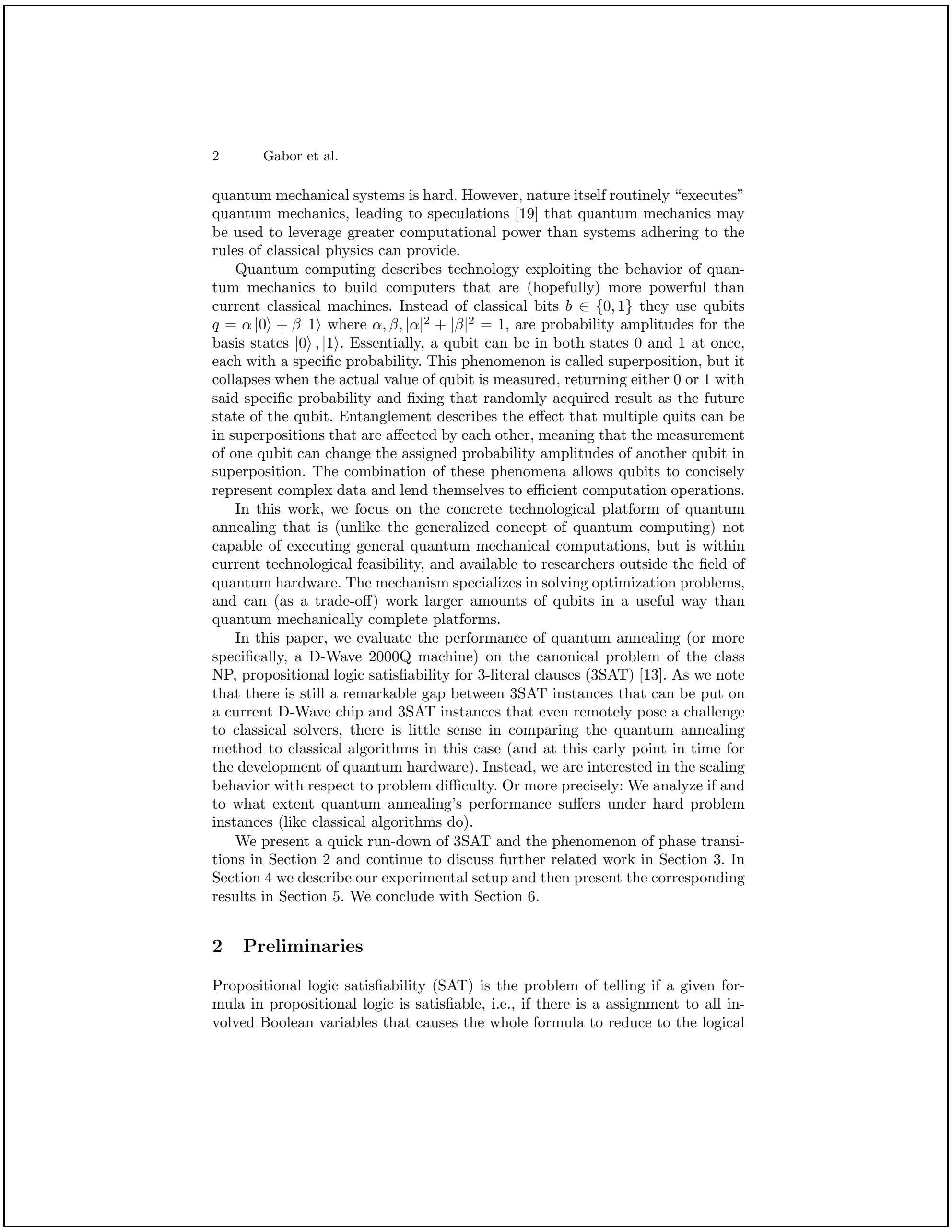
Quantum computers are an emerging technology and still subject to frequent new developments. Eventually, the utilization of intricate physical phenomena like superposition and entanglement is conjectured to provide an advantage in computational power over purely classical computers. As of now, however, the first practical breakthrough application for quantum computers is still sought for. But new results on the behavior of quantum programs in comparison to their classical counterparts are reported on a daily basis.

Research in that area has cast an eye on the complexity class NP: It contains problems that are traditionally (and at the current state of knowledge regarding the P vs. NP problem) conjectured to produce instances too hard for classical computers to solve exactly and deterministically within practical time constraints. Still, problem instances of NP are also easy enough that they can be executed efficiently on a (hypothetical) non-deterministic computer.

The notion of computational complexity is based on classical computation in the sense of using classical mechanics to describe and perform automated computations. In particular, it is known that in this model of computation, simulating

This is a pre-print of an article to be published in *Feld S., Linnhoff-Popien C. (eds)*, Quantum Technology and Optimization Problems (QTOP 2019), Lecture Notes in Computer Science, vol 11413, Springer, 2019. The final authenticated version is available online at DOI: http://doi.org/10.1007/978-3-030-14082-3

Taken from original publication: Thomas Gabor, Sebastian Zielinski, Sebastian Feld, Christoph Roch, Christian Seidel, Florian Neukart, Isabella Galter, Wolfgang Mauerer, and Claudia Linnhoff-Popien. Assessing solution quality of 3SAT on a quantum annealing platform. In *International Workshop on Quantum Technology and Optimization Problems*. Springer, 2019



quantum mechanical systems is hard. However, nature itself routinely "executes" quantum mechanics, leading to speculations [19] that quantum mechanics may be used to leverage greater computational power than systems adhering to the rules of classical physics can provide.

Quantum computing describes technology exploiting the behavior of quantum mechanics to build computers that are (hopefully) more powerful than current classical machines. Instead of classical bits $b \in \{0,1\}$ they use qubits $q = \alpha\left|0\right\rangle + \beta\left|1\right\rangle$ where $\alpha, \beta, |\alpha|^2 + |\beta|^2 = 1$, are probability amplitudes for the basis states $\left|0\right\rangle, \left|1\right\rangle$. Essentially, a qubit can be in both states 0 and 1 at once, each with a specific probability. This phenomenon is called superposition, but it collapses when the actual value of qubit is measured, returning either 0 or 1 with said specific probability and fixing that randomly acquired result as the future state of the qubit. Entanglement describes the effect that multiple quits can be in superpositions that are affected by each other, meaning that the measurement of one qubit can change the assigned probability amplitudes of another qubit in superposition. The combination of these phenomena allows qubits to concisely represent complex data and lend themselves to efficient computation operations.

In this work, we focus on the concrete technological platform of quantum annealing that is (unlike the generalized concept of quantum computing) not capable of executing general quantum mechanical computations, but is within current technological feasibility, and available to researchers outside the field of quantum hardware. The mechanism specializes in solving optimization problems, and can (as a trade-off) work larger amounts of qubits in a useful way than quantum mechanically complete platforms.

In this paper, we evaluate the performance of quantum annealing (or more specifically, a D-Wave 2000Q machine) on the canonical problem of the class NP, propositional logic satisfiability for 3-literal clauses (3SAT) [13]. As we note that there is still a remarkable gap between 3SAT instances that can be put on a current D-Wave chip and 3SAT instances that even remotely pose a challenge to classical solvers, there is little sense in comparing the quantum annealing method to classical algorithms in this case (and at this early point in time for the development of quantum hardware). Instead, we are interested in the scaling behavior with respect to problem difficulty. Or more precisely: We analyze if and to what extent quantum annealing's performance suffers under hard problem instances (like classical algorithms do).

We present a quick run-down of 3SAT and the phenomenon of phase transitions in Section 2 and continue to discuss further related work in Section 3. In Section 4 we describe our experimental setup and then present the corresponding results in Section 5. We conclude with Section 6.

2 Preliminaries

Propositional logic satisfiability (SAT) is the problem of telling if a given formula in propositional logic is satisfiable, i.e., if there is a assignment to all involved Boolean variables that causes the whole formula to reduce to the logical

Taken from original publication: Thomas Gabor, Sebastian Zielinski, Sebastian Feld, Christoph Roch, Christian Seidel, Florian Neukart, Isabella Galter, Wolfgang Mauerer, and Claudia Linnhoff-Popien. Assessing solution quality of 3SAT on a quantum annealing platform. In *International Workshop on Quantum Technology and Optimization Problems*. Springer, 2019

value *True*. As such, the problem occurs at every application involved complex constraints or reasoning, like (software) product lines, the tracing of software dependencies or formal methods.

It can be trivially shown that (when introducing a linear amount of new variables) all SAT problems can be reduced to a specific type of SAT problem called 3SAT, where the input propositional logic formula has to be in conjunctive normal form with all of the disjunctions containing exactly three literals.

For example, the formula $\Psi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$ is in 3SAT form and is satisfiable because the assignment $(x_1 \mapsto \mathit{True}, x_2 \mapsto \mathit{True}, x_2 \mapsto \mathit{True})$ causes the formula to reduce to *True*. The formula $\Phi = (x_1 \vee x_1 \vee x_1) \wedge (\neg x_1 \vee \neg x_1 \vee \neg x_1)$ is also in 3SAT form but is not satisfiable.

Definition (3SAT) A 3SAT instance with m clauses and n variables is given as a list of clauses $(c_k)_{0 \leq k \leq m-1}$ of the form $c_k = (l_{3k} \vee l_{3k+1} \vee l_{3k+2})$ and a list of variables $(v_j)_{0 \leq j \leq n-1}$ so that l_i is a literal of the form $l_i \in \bigcup_{0 \leq j \leq n-1} \{v_j, \neg v_j\}$. A given 3SAT instance is *satisfiable* iff there exists a variable assignment $(v_j \mapsto b_j)_{0 \leq j \leq n-1}$ with $b_j \in \{\mathit{True}, \mathit{False}\}$ so that $\bigwedge_{0 \leq k \leq m-1} c_k$ reduces to *True* when interpreting all logical operators as is common. The problem of deciding whether a given 3SAT instance is satisfiable is called 3SAT.

3SAT is of special importance to complexity theory as it was the first problem which was shown to be NP-complete [13]. This means that every problem in NP can be reduced to 3SAT in polynomial time. It follows that any means to solve 3SAT efficiently would thus give rise to efficient solutions for any problem in NP like graph coloring, travelling salesman or bin packing.

Despite the fact that for NP-complete problems in general no algorithm is known that can solve all problem instances of a problem efficiently (i.e., in polynomial time), it is within the scope of knowledge that "average" problem instances of many NP-complete problems, including 3SAT, are easy to solve [9]. In Ref. [36] this characteristic is described with a phase transition. The boundary of the phase transition divides the problem space into two regions. In one region, a solution can be found relatively easily, because the solution density for these problems is high, whereas in the other region, it is very unlikely that problems can contain a correct solution at all. Problems that are very difficult to solve are located directly at this phase boundary [9].

It can be observed that, with randomly generated 3SAT instances, the probability of finding a correct solution decreases abruptly when the ratio of clauses to variables $\alpha = m/n$ exceeds a critical value of α_c [35]. According to [34] this critical point is $\alpha_c \approx 4.267$ for randomly generated 3SAT instances. In the surrounding area of the critical point, finding a solution (i.e., deciding if the instance is satisfiable) is algorithmically complex. Figure 1 illustrates this phenomenon.

To assess the solution quality of randomly generated 3SAT instances we generate instances in every complexity region. The results are discussed in Section 5.

Taken from original publication: Thomas Gabor, Sebastian Zielinski, Sebastian Feld, Christoph Roch, Christian Seidel, Florian Neukart, Isabella Galter, Wolfgang Mauerer, and Claudia Linnhoff-Popien. Assessing solution quality of 3SAT on a quantum annealing platform. In *International Workshop on Quantum Technology and Optimization Problems*. Springer, 2019

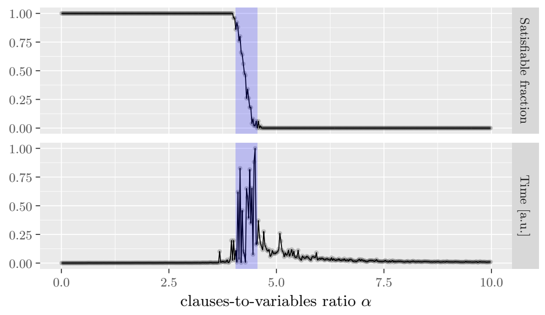


Fig. 1. Phase transition of SAT. The bottom plot shows the computational time required to determine satisfiability of randomly chosen 3SAT instances with specific a clauses-to-variables ratio α on a standard solver. The area around the critical point $\alpha_c \approx 4.267$ is shaded in blue.
The upper portion shows the probability that instances with a particular ratio α are solvable. In the region around the critical point, it is hard to determine whether a problem instance can be fulfilled with a concrete allocation or not.

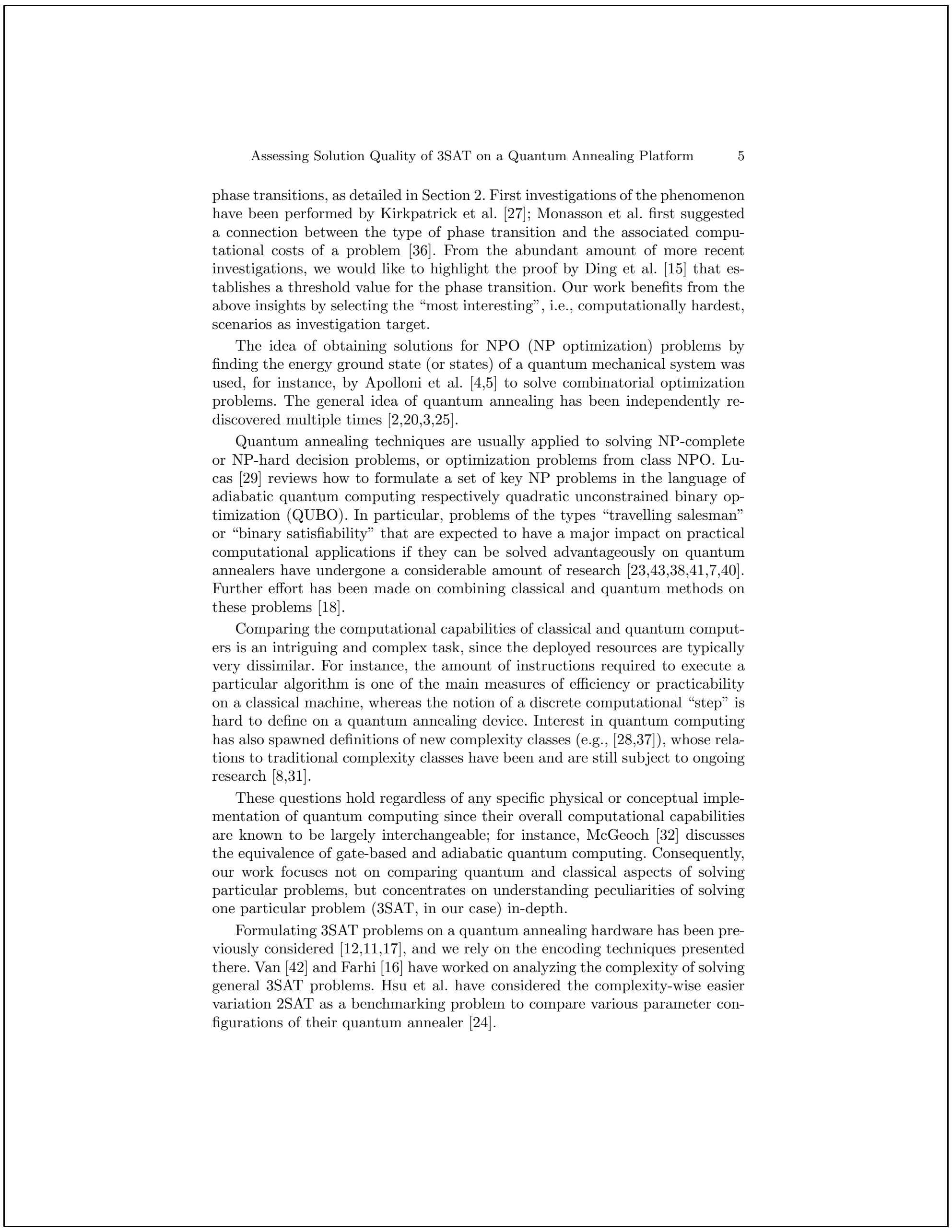
3 Related Work

It is one of the cornerstones of complexity theory that solving NP-complete or even NP-hard decision problems is strongly believed to be not efficiently possible [13,39]. Any NP-complete problem can also be cast as an optimization problem, which allows for employing well-known optimization algorithms to find approximate solutions—typical methods include tabu search [22,21] and simulated annealing [26,10]. Countless other efficient approximation methods, together with an elaborate taxonomy on approximation quality (how much does a given solution differ from a known global optimum?) and computational effort (how many time steps are required until an approximate solution that satisfies given quality goals is available?), have been devised [6].

Some problem (knapsack, e.g.) exhibit favorable properties when cast as an optimization problem. The latter is a member of the complexity class FPTAS (fully polynomial-time approximation scheme), which means that a solution with distance $1 + \epsilon$ (of course, $\epsilon > 0$) from an optimal solution can be determined in polynomial time in both, input size n and inverse approximation quality $1/\epsilon$ [10].

An intriguing connection that has received substantial attraction exists between (computational) NP-complete problems and the (physical) concept of

Taken from original publication: Thomas Gabor, Sebastian Zielinski, Sebastian Feld, Christoph Roch, Christian Seidel, Florian Neukart, Isabella Galter, Wolfgang Mauerer, and Claudia Linnhoff-Popien. Assessing solution quality of 3SAT on a quantum annealing platform. In *International Workshop on Quantum Technology and Optimization Problems*. Springer, 2019



phase transitions, as detailed in Section 2. First investigations of the phenomenon have been performed by Kirkpatrick et al. [27]; Monasson et al. first suggested a connection between the type of phase transition and the associated computational costs of a problem [36]. From the abundant amount of more recent investigations, we would like to highlight the proof by Ding et al. [15] that establishes a threshold value for the phase transition. Our work benefits from the above insights by selecting the "most interesting", i.e., computationally hardest, scenarios as investigation target.

The idea of obtaining solutions for NPO (NP optimization) problems by finding the energy ground state (or states) of a quantum mechanical system was used, for instance, by Apolloni et al. [4,5] to solve combinatorial optimization problems. The general idea of quantum annealing has been independently rediscovered multiple times [2,20,3,25].

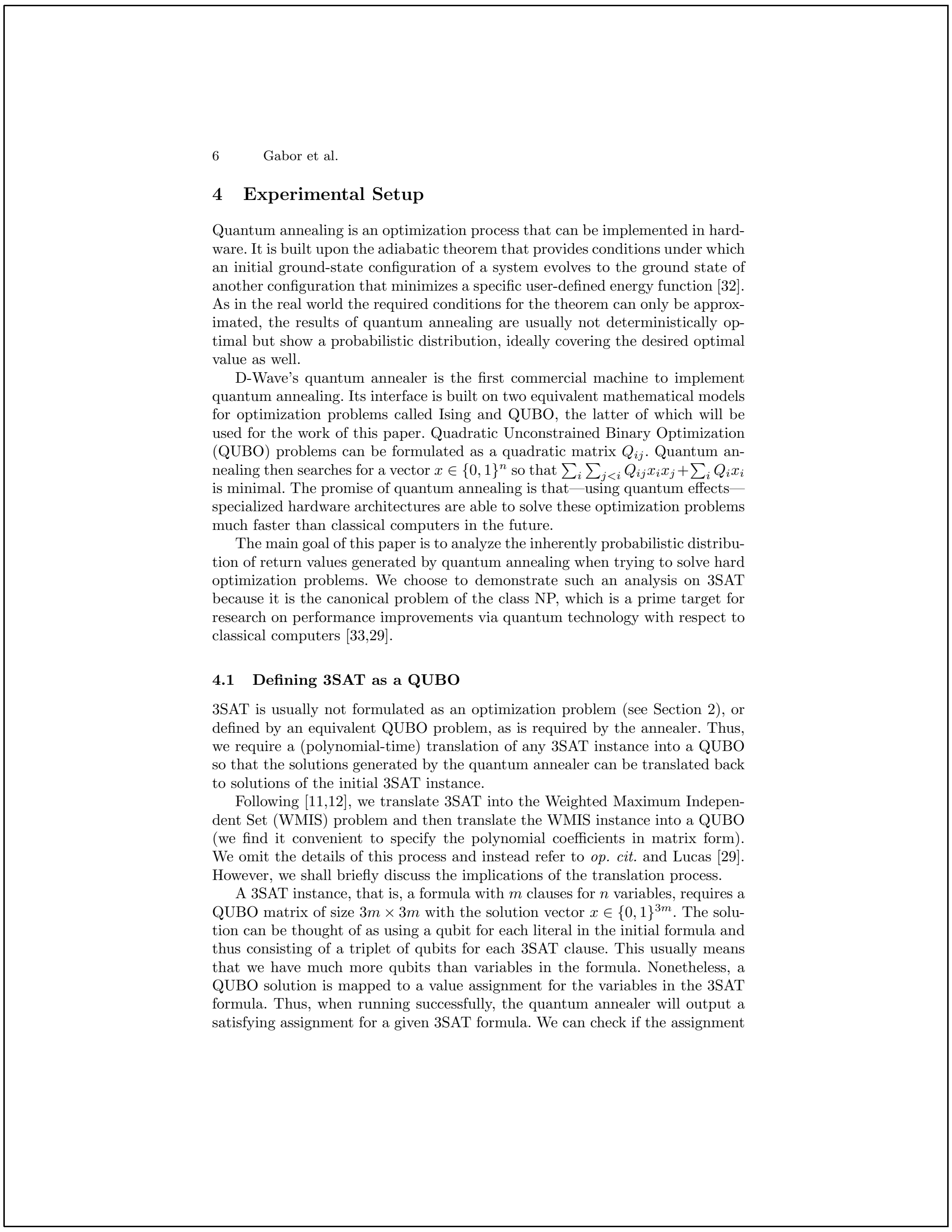
Quantum annealing techniques are usually applied to solving NP-complete or NP-hard decision problems, or optimization problems from class NPO. Lucas [29] reviews how to formulate a set of key NP problems in the language of adiabatic quantum computing respectively quadratic unconstrained binary optimization (QUBO). In particular, problems of the types "travelling salesman" or "binary satisfiability" that are expected to have a major impact on practical computational applications if they can be solved advantageously on quantum annealers have undergone a considerable amount of research [23,43,38,41,7,40]. Further effort has been made on combining classical and quantum methods on these problems [18].

Comparing the computational capabilities of classical and quantum computers is an intriguing and complex task, since the deployed resources are typically very dissimilar. For instance, the amount of instructions required to execute a particular algorithm is one of the main measures of efficiency or practicability on a classical machine, whereas the notion of a discrete computational "step" is hard to define on a quantum annealing device. Interest in quantum computing has also spawned definitions of new complexity classes (e.g., [28,37]), whose relations to traditional complexity classes have been and are still subject to ongoing research [8,31].

These questions hold regardless of any specific physical or conceptual implementation of quantum computing since their overall computational capabilities are known to be largely interchangeable; for instance, McGeoch [32] discusses the equivalence of gate-based and adiabatic quantum computing. Consequently, our work focuses not on comparing quantum and classical aspects of solving particular problems, but concentrates on understanding peculiarities of solving one particular problem (3SAT, in our case) in-depth.

Formulating 3SAT problems on a quantum annealing hardware has been previously considered [12,11,17], and we rely on the encoding techniques presented there. Van [42] and Farhi [16] have worked on analyzing the complexity of solving general 3SAT problems. Hsu et al. have considered the complexity-wise easier variation 2SAT as a benchmarking problem to compare various parameter configurations of their quantum annealer [24].

Taken from original publication: Thomas Gabor, Sebastian Zielinski, Sebastian Feld, Christoph Roch, Christian Seidel, Florian Neukart, Isabella Galter, Wolfgang Mauerer, and Claudia Linnhoff-Popien. Assessing solution quality of 3SAT on a quantum annealing platform. In *International Workshop on Quantum Technology and Optimization Problems*. Springer, 2019



4 Experimental Setup

Quantum annealing is an optimization process that can be implemented in hardware. It is built upon the adiabatic theorem that provides conditions under which an initial ground-state configuration of a system evolves to the ground state of another configuration that minimizes a specific user-defined energy function [32]. As in the real world the required conditions for the theorem can only be approximated, the results of quantum annealing are usually not deterministically optimal but show a probabilistic distribution, ideally covering the desired optimal value as well.

D-Wave's quantum annealer is the first commercial machine to implement quantum annealing. Its interface is built on two equivalent mathematical models for optimization problems called Ising and QUBO, the latter of which will be used for the work of this paper. Quadratic Unconstrained Binary Optimization (QUBO) problems can be formulated as a quadratic matrix Q_{ij}. Quantum annealing then searches for a vector $x \in \{0,1\}^n$ so that $\sum_i \sum_{j<i} Q_{ij} x_i x_j + \sum_i Q_i x_i$ is minimal. The promise of quantum annealing is that—using quantum effects—specialized hardware architectures are able to solve these optimization problems much faster than classical computers in the future.

The main goal of this paper is to analyze the inherently probabilistic distribution of return values generated by quantum annealing when trying to solve hard optimization problems. We choose to demonstrate such an analysis on 3SAT because it is the canonical problem of the class NP, which is a prime target for research on performance improvements via quantum technology with respect to classical computers [33,29].

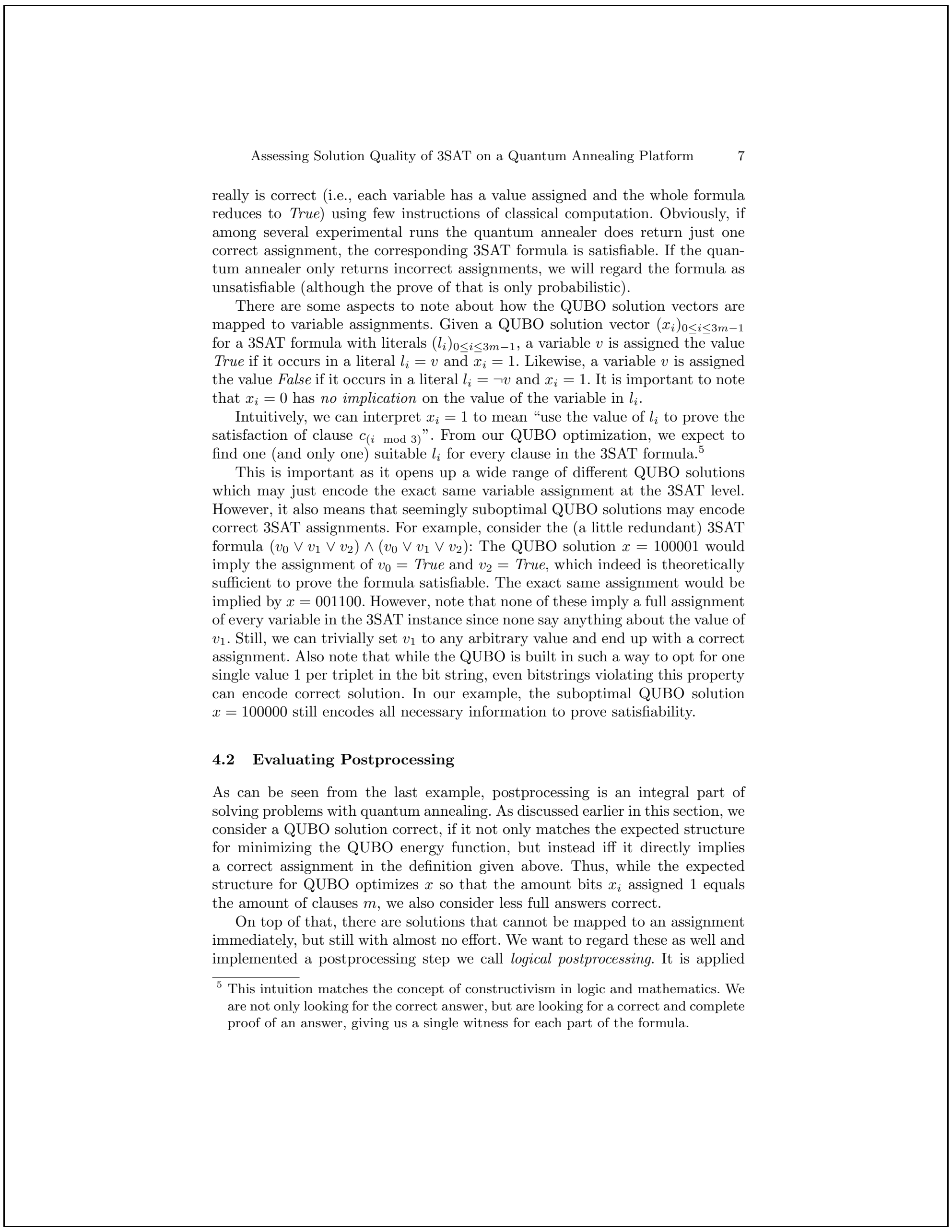
4.1 Defining 3SAT as a QUBO

3SAT is usually not formulated as an optimization problem (see Section 2), or defined by an equivalent QUBO problem, as is required by the annealer. Thus, we require a (polynomial-time) translation of any 3SAT instance into a QUBO so that the solutions generated by the quantum annealer can be translated back to solutions of the initial 3SAT instance.

Following [11,12], we translate 3SAT into the Weighted Maximum Independent Set (WMIS) problem and then translate the WMIS instance into a QUBO (we find it convenient to specify the polynomial coefficients in matrix form). We omit the details of this process and instead refer to *op. cit.* and Lucas [29]. However, we shall briefly discuss the implications of the translation process.

A 3SAT instance, that is, a formula with m clauses for n variables, requires a QUBO matrix of size $3m \times 3m$ with the solution vector $x \in \{0,1\}^{3m}$. The solution can be thought of as using a qubit for each literal in the initial formula and thus consisting of a triplet of qubits for each 3SAT clause. This usually means that we have much more qubits than variables in the formula. Nonetheless, a QUBO solution is mapped to a value assignment for the variables in the 3SAT formula. Thus, when running successfully, the quantum annealer will output a satisfying assignment for a given 3SAT formula. We can check if the assignment

Taken from original publication: Thomas Gabor, Sebastian Zielinski, Sebastian Feld, Christoph Roch, Christian Seidel, Florian Neukart, Isabella Galter, Wolfgang Mauerer, and Claudia Linnhoff-Popien. Assessing solution quality of 3SAT on a quantum annealing platform. In *International Workshop on Quantum Technology and Optimization Problems.* Springer, 2019



really is correct (i.e., each variable has a value assigned and the whole formula reduces to *True*) using few instructions of classical computation. Obviously, if among several experimental runs the quantum annealer does return just one correct assignment, the corresponding 3SAT formula is satisfiable. If the quantum annealer only returns incorrect assignments, we will regard the formula as unsatisfiable (although the prove of that is only probabilistic).

There are some aspects to note about how the QUBO solution vectors are mapped to variable assignments. Given a QUBO solution vector $(x_i)_{0 \leq i \leq 3m-1}$ for a 3SAT formula with literals $(l_i)_{0 \leq i \leq 3m-1}$, a variable v is assigned the value *True* if it occurs in a literal $l_i = v$ and $x_i = 1$. Likewise, a variable v is assigned the value *False* if it occurs in a literal $l_i = \neg v$ and $x_i = 1$. It is important to note that $x_i = 0$ has *no implication* on the value of the variable in l_i.

Intuitively, we can interpret $x_i = 1$ to mean "use the value of l_i to prove the satisfaction of clause $c_{(i \mod 3)}$". From our QUBO optimization, we expect to find one (and only one) suitable l_i for every clause in the 3SAT formula.[5]

This is important as it opens up a wide range of different QUBO solutions which may just encode the exact same variable assignment at the 3SAT level. However, it also means that seemingly suboptimal QUBO solutions may encode correct 3SAT assignments. For example, consider the (a little redundant) 3SAT formula $(v_0 \lor v_1 \lor v_2) \land (v_0 \lor v_1 \lor v_2)$: The QUBO solution $x = 100001$ would imply the assignment of $v_0 =$ *True* and $v_2 =$ *True*, which indeed is theoretically sufficient to prove the formula satisfiable. The exact same assignment would be implied by $x = 001100$. However, note that none of these imply a full assignment of every variable in the 3SAT instance since none say anything about the value of v_1. Still, we can trivially set v_1 to any arbitrary value and end up with a correct assignment. Also note that while the QUBO is built in such a way to opt for one single value 1 per triplet in the bit string, even bitstrings violating this property can encode correct solution. In our example, the suboptimal QUBO solution $x = 100000$ still encodes all necessary information to prove satisfiability.

4.2 Evaluating Postprocessing

As can be seen from the last example, postprocessing is an integral part of solving problems with quantum annealing. As discussed earlier in this section, we consider a QUBO solution correct, if it not only matches the expected structure for minimizing the QUBO energy function, but instead iff it directly implies a correct assignment in the definition given above. Thus, while the expected structure for QUBO optimizes x so that the amount bits x_i assigned 1 equals the amount of clauses m, we also consider less full answers correct.

On top of that, there are solutions that cannot be mapped to an assignment immediately, but still with almost no effort. We want to regard these as well and implemented a postprocessing step we call *logical postprocessing.* It is applied

[5] This intuition matches the concept of constructivism in logic and mathematics. We are not only looking for the correct answer, but are looking for a correct and complete proof of an answer, giving us a single witness for each part of the formula.

Taken from original publication: Thomas Gabor, Sebastian Zielinski, Sebastian Feld, Christoph Roch, Christian Seidel, Florian Neukart, Isabella Galter, Wolfgang Mauerer, and Claudia Linnhoff-Popien. Assessing solution quality of 3SAT on a quantum annealing platform. In *International Workshop on Quantum Technology and Optimization Problems.* Springer, 2019

whenever none of the qubits corresponding to a single clause c_k are set to 1 by the quantum annealer and the respective QUBO solution is not already correct. In that case, we iterate through all literals l_i in that clause c_k and check if we could set $x_i = 1$ without contradicting any other assignment made within x. If we find such an l_i, we set $x_i = 1$ and return the altered bitstring x.

The software platform provided by D-Wave to use the quantum annealer already offers integrated postprocessing methods as well, which we will also empirically show to be more powerful than logical postprocessing in the following Section 5. Again, for greater detail we refer to the D-Wave documentation on that matter [14]. At a glance, the employed postprocessing method splits the QUBO matrix into several subproblems, tries to optimize these locally, and then integrates that local solution into the complete solution if it yields an improvement. We call this method *D-Wave postprocessing*.

To evaluate the solution quality regarding 3SAT, we employ both methods. The goal is to assess the expected quality on a 3SAT-to-3SAT level, that is, we measure how well we can solve the given 3SAT instance and regard the translation to and from QUBO as a mere technical problem that is not of interest for this paper.

5 Evaluation

To assess the solution quality of 3SAT on a quantum annealing platform, using the previously discussed method of encoding 3SAT problems, we ran several experiments on a D-Wave 2000Q system. Using ToughSAT[6] we generated 3SAT instances of various difficulty (i.e., with various values for α). However, as discussed in Section 2, for $|\alpha - 4.2| \gg 0$ problem instances become very easy to solve. We observed that effect on the quantum annealer as well, since all of these instances were easily solved on the D-Wave machine. Thus, for the remainder of this section, we focus on hard instances (approximated by $\alpha = 4.2$) to assess solution quality in the interesting problem domain.

Experiments have shown that using the standard embedding tools delivered with the D-Wave platform, we can only reliably find a working embedding on the D-Wave 2000Q chip for 3SAT instances with at most 42 clauses [1]. To maintain $\alpha \approx 4.2$, the generated 3SAT instances contain 10 different variables. We only assess solution quality for 3SAT instances that are satisfiable, but do not provide this information to the solver.

Figure 2 shows the result distribution of these runs on the D-Wave machine. On the x-axis, we sorted the returned results according to the bits that have been assigned the value 1 or *True*. As discussed in Section 4 the optimal solution is supposed to set one bit for each clause, i.e., is supposed to contain 42 bits set to *True*. However, as there are only 10 different variables, there theoretically exist answers that only set 10 bits but that still map to a complete and valid solution for the given 3SAT instance. From Figure 2 we can see that some of

[6] `https://toughsat.appspot.com/`

Taken from original publication: Thomas Gabor, Sebastian Zielinski, Sebastian Feld, Christoph Roch, Christian Seidel, Florian Neukart, Isabella Galter, Wolfgang Mauerer, and Claudia Linnhoff-Popien. Assessing solution quality of 3SAT on a quantum annealing platform. In *International Workshop on Quantum Technology and Optimization Problems.* Springer, 2019

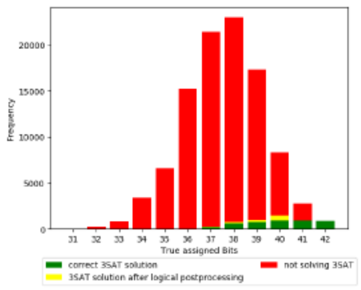


Fig. 2. Distribution of correct (green) and incorrect (red) answers returned by the quantum annealer *without D-WAVE postprocessing.* Answers that can trivially be transformed into valid answers using logical postprocessing are marked in yellow. The plot shows 100,000 answers in total for 100 different hard 3SAT instances ($\alpha \approx 4.2$).

these solution are found for bitcounts starting from 37 through 41. Interestingly, the complete range of answers gathered seems to follow a distribution centered around 37 or 38 and no answers with more than 42 bits are returned. This means that the constraint of never setting multiple bits per clause is fully respected in the evaluation of our QUBO matrix. It is important to note that although there are 5,283 correct solutions in total, these are only distributed across 24 of the 100 randomly generated problem instances. Thus, most of them have not been solved at all.

Furthermore, we applied the logical postprocessing described in Section 4 to the incorrect answers in Figure 2. However, it shows little improvement on the total amount of correct answers collected. We expect the postprocessing method delivered with the D-Wave software package to be more powerful as it runs local search along more axes of the solution space than the logical postprocessing does. So we ran the complete evaluation experiment again, only this time turning on the integrated postprocessing. The results are shown in Figure 3.

We observed that the D-Wave postprocessing managed to optimize all correct but "incomplete" answers, mapping them to a solution with 42 bits assigned the value *True*. Out of the 100,000 queries, this yielded 25,142 correct answers. Moreover, these correct answers span 99 of the 100 randomly generated 3SAT instances so that we consider the problem solved. Effectively, this shows that quantum annealing does suffer from a breakdown in expected solution quality at the point of the phase transition in the 3SAT problem. In comparison to the immense decrease in performance seen in classical solvers (cf. Section 2), a drop to around 25% precision appears rather desirable, though. A quick example: To

Taken from original publication: Thomas Gabor, Sebastian Zielinski, Sebastian Feld, Christoph Roch, Christian Seidel, Florian Neukart, Isabella Galter, Wolfgang Mauerer, and Claudia Linnhoff-Popien. Assessing solution quality of 3SAT on a quantum annealing platform. In *International Workshop on Quantum Technology and Optimization Problems*. Springer, 2019

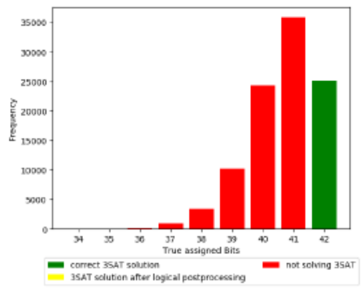


Fig. 3. Distribution of correct (green) and incorrect (red) answers returned by the quantum annealer *using D-WAVE postprocessing*. Answers that can trivially be transformed into valid answers using logical postprocessing are marked in yellow. The plot shows 100,000 answers in total for 100 different hard 3SAT instances ($\alpha \approx 4.2$).

achieve a $1 - 10^{-12}$ confidence of returning the correct answer our experimental setup requires around 97 queries. At a glance, that scaling factor with respect to problem difficulty is much better than what is observed for classical algorithms: For example, in the data used for Figure 1 we observed performance decrease up to one order of magnitude larger. It is important to note, however, that these experiments were performed for problem instances so small that their evaluation does not pose a challenge to classical processors at all, i.e., below the point of reasonable performance metrics. Thus, these results only proof relevant to practical applications if they scale with future versions of quantum annealing hardware that can tackle much larger problem instances.

So far, we have not discerned between different correct solutions. We were content as long as the algorithm returned but one. However, for the user it is interesting to know if he or she will receive the same solution with every answer or an even distribution across the complete solution space. Our experiments show that when a lot of correct solutions are found for a certain problem instance, there are cases where we can see a clear bias towards a specific solution variant. Figure 4 shows the distributions of specific solutions. While some formulae seem to yield rather narrow distributions over the different possible answers, others definitely seem to have a bias towards certain solutions. However, the former also tend to have relatively smaller sample sizes as there are less solutions in total to consider. Further investigation could still reveal a distinctive distribution in these cases as well. Thus, we consider this behavior of the quantum annealer to be roughly in line with the findings of [30], who show an exponential bias in ground-state sampling of a quantum annealer.

Taken from original publication: Thomas Gabor, Sebastian Zielinski, Sebastian Feld, Christoph Roch, Christian Seidel, Florian Neukart, Isabella Galter, Wolfgang Mauerer, and Claudia Linnhoff-Popien. Assessing solution quality of 3SAT on a quantum annealing platform. In *International Workshop on Quantum Technology and Optimization Problems*. Springer, 2019

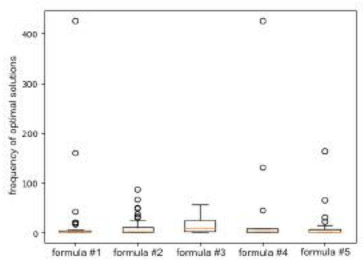


Fig. 4. Frequency of occurrence of different solutions for 5 formulae with many returned solutions. While most solutions are found once or just a few times, there are specific solutions that are found much more often.

6 Conclusion

We have shown that problem difficulty of 3SAT instances also affects the performance of quantum annealing as it does for classical algorithms. However, bound by the nature of both approaches, the effects are quite different with complete classical algorithms showing longer runtimes and quantum annealing showing less precision. A first quantification of that loss of precision suggests that it may not be too detrimental and comparatively easy to deal with. However, because of to the maximum available chip size for quantum annealing hardware at the moment, no large-scale test could be performed. No real assumptions on the scaling of this phenomenon (and thus the eventual real-world benefit) can be made yet.

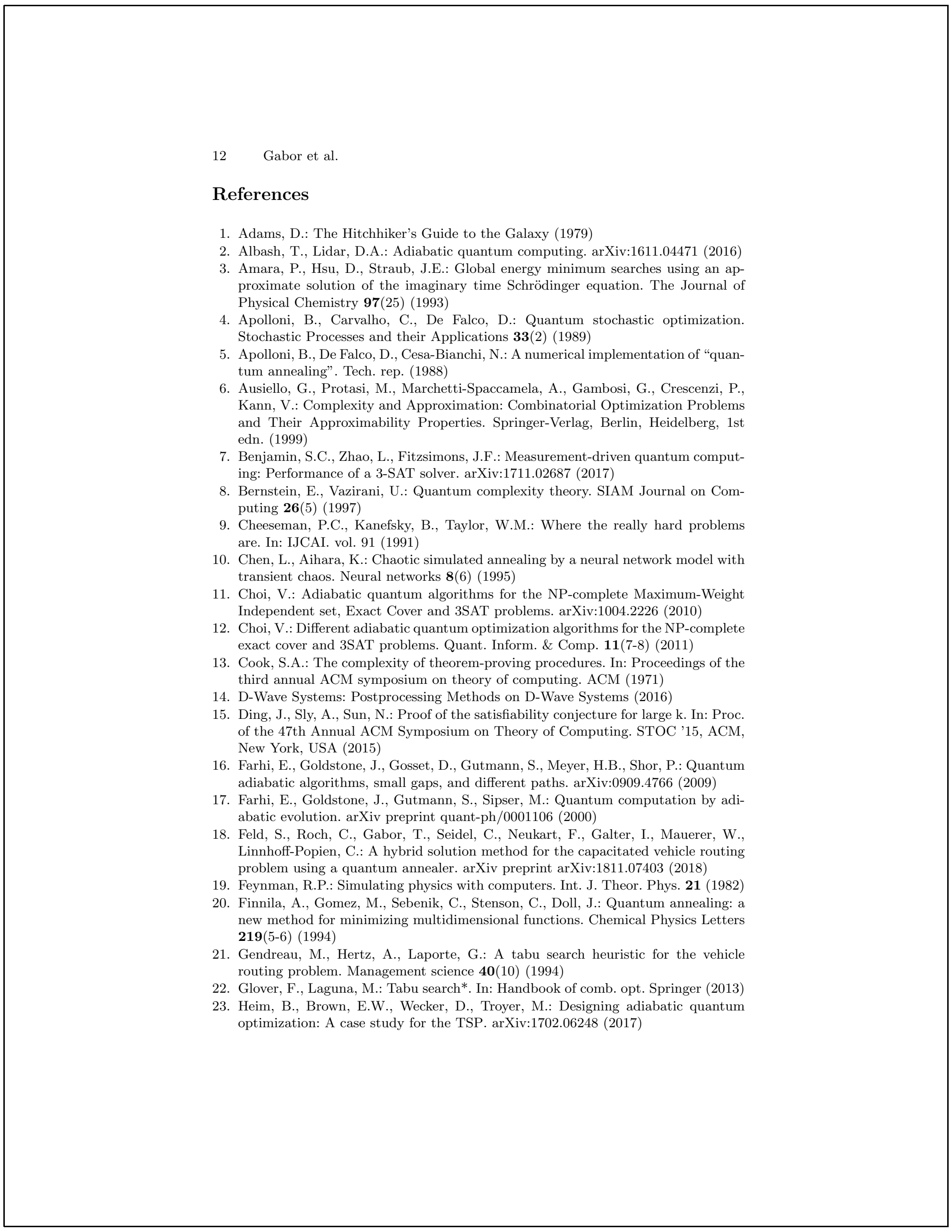
Our results suggest there are cases where single solutions from a set of equally optimal solutions are much more likely to be returned than others. This observation is in line with other literature on the results of quantum annealing. However, it is interesting to note that it translates into the original problem space of 3SAT.

The observed results will gain more practical relevance with larger chip sizes for quantum annealers. We thus suggest to perform these and/or similar tests for future editions of quantum annealing hardware. If the effects persist, they can indicate a substantial advantage of quantum hardware over other known approaches for solving NP-complete problems.

Acknowledgement

Research was funded by Volkswagen Group, department Group IT.

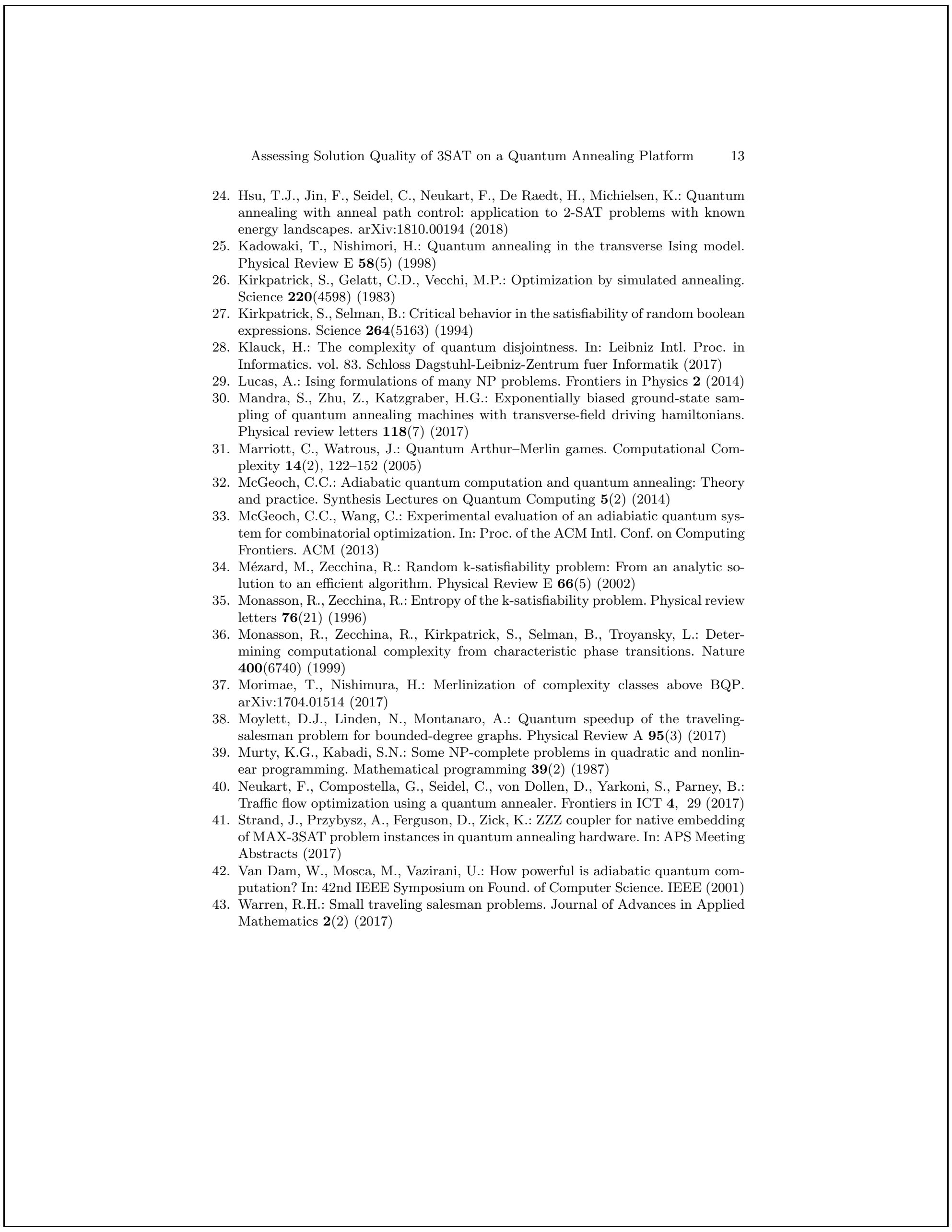
Taken from original publication: Thomas Gabor, Sebastian Zielinski, Sebastian Feld, Christoph Roch, Christian Seidel, Florian Neukart, Isabella Galter, Wolfgang Mauerer, and Claudia Linnhoff-Popien. Assessing solution quality of 3SAT on a quantum annealing platform. In *International Workshop on Quantum Technology and Optimization Problems*. Springer, 2019



References

1. Adams, D.: The Hitchhiker's Guide to the Galaxy (1979)
2. Albash, T., Lidar, D.A.: Adiabatic quantum computing. arXiv:1611.04471 (2016)
3. Amara, P., Hsu, D., Straub, J.E.: Global energy minimum searches using an approximate solution of the imaginary time Schrödinger equation. The Journal of Physical Chemistry **97**(25) (1993)
4. Apolloni, B., Carvalho, C., De Falco, D.: Quantum stochastic optimization. Stochastic Processes and their Applications **33**(2) (1989)
5. Apolloni, B., De Falco, D., Cesa-Bianchi, N.: A numerical implementation of "quantum annealing". Tech. rep. (1988)
6. Ausiello, G., Protasi, M., Marchetti-Spaccamela, A., Gambosi, G., Crescenzi, P., Kann, V.: Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties. Springer-Verlag, Berlin, Heidelberg, 1st edn. (1999)
7. Benjamin, S.C., Zhao, L., Fitzsimons, J.F.: Measurement-driven quantum computing: Performance of a 3-SAT solver. arXiv:1711.02687 (2017)
8. Bernstein, E., Vazirani, U.: Quantum complexity theory. SIAM Journal on Computing **26**(5) (1997)
9. Cheeseman, P.C., Kanefsky, B., Taylor, W.M.: Where the really hard problems are. In: IJCAI. vol. 91 (1991)
10. Chen, L., Aihara, K.: Chaotic simulated annealing by a neural network model with transient chaos. Neural networks **8**(6) (1995)
11. Choi, V.: Adiabatic quantum algorithms for the NP-complete Maximum-Weight Independent set, Exact Cover and 3SAT problems. arXiv:1004.2226 (2010)
12. Choi, V.: Different adiabatic quantum optimization algorithms for the NP-complete exact cover and 3SAT problems. Quant. Inform. & Comp. **11**(7-8) (2011)
13. Cook, S.A.: The complexity of theorem-proving procedures. In: Proceedings of the third annual ACM symposium on theory of computing. ACM (1971)
14. D-Wave Systems: Postprocessing Methods on D-Wave Systems (2016)
15. Ding, J., Sly, A., Sun, N.: Proof of the satisfiability conjecture for large k. In: Proc. of the 47th Annual ACM Symposium on Theory of Computing. STOC '15, ACM, New York, USA (2015)
16. Farhi, E., Goldstone, J., Gosset, D., Gutmann, S., Meyer, H.B., Shor, P.: Quantum adiabatic algorithms, small gaps, and different paths. arXiv:0909.4766 (2009)
17. Farhi, E., Goldstone, J., Gutmann, S., Sipser, M.: Quantum computation by adiabatic evolution. arXiv preprint quant-ph/0001106 (2000)
18. Feld, S., Roch, C., Gabor, T., Seidel, C., Neukart, F., Galter, I., Mauerer, W., Linnhoff-Popien, C.: A hybrid solution method for the capacitated vehicle routing problem using a quantum annealer. arXiv preprint arXiv:1811.07403 (2018)
19. Feynman, R.P.: Simulating physics with computers. Int. J. Theor. Phys. **21** (1982)
20. Finnila, A., Gomez, M., Sebenik, C., Stenson, C., Doll, J.: Quantum annealing: a new method for minimizing multidimensional functions. Chemical Physics Letters **219**(5-6) (1994)
21. Gendreau, M., Hertz, A., Laporte, G.: A tabu search heuristic for the vehicle routing problem. Management science **40**(10) (1994)
22. Glover, F., Laguna, M.: Tabu search*. In: Handbook of comb. opt. Springer (2013)
23. Heim, B., Brown, E.W., Wecker, D., Troyer, M.: Designing adiabatic quantum optimization: A case study for the TSP. arXiv:1702.06248 (2017)

Taken from original publication: Thomas Gabor, Sebastian Zielinski, Sebastian Feld, Christoph Roch, Christian Seidel, Florian Neukart, Isabella Galter, Wolfgang Mauerer, and Claudia Linnhoff-Popien. Assessing solution quality of 3SAT on a quantum annealing platform. In *International Workshop on Quantum Technology and Optimization Problems*. Springer, 2019



24. Hsu, T.J., Jin, F., Seidel, C., Neukart, F., De Raedt, H., Michielsen, K.: Quantum annealing with anneal path control: application to 2-SAT problems with known energy landscapes. arXiv:1810.00194 (2018)
25. Kadowaki, T., Nishimori, H.: Quantum annealing in the transverse Ising model. Physical Review E **58**(5) (1998)
26. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science **220**(4598) (1983)
27. Kirkpatrick, S., Selman, B.: Critical behavior in the satisfiability of random boolean expressions. Science **264**(5163) (1994)
28. Klauck, H.: The complexity of quantum disjointness. In: Leibniz Intl. Proc. in Informatics. vol. 83. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2017)
29. Lucas, A.: Ising formulations of many NP problems. Frontiers in Physics **2** (2014)
30. Mandra, S., Zhu, Z., Katzgraber, H.G.: Exponentially biased ground-state sampling of quantum annealing machines with transverse-field driving hamiltonians. Physical review letters **118**(7) (2017)
31. Marriott, C., Watrous, J.: Quantum Arthur–Merlin games. Computational Complexity **14**(2), 122–152 (2005)
32. McGeoch, C.C.: Adiabatic quantum computation and quantum annealing: Theory and practice. Synthesis Lectures on Quantum Computing **5**(2) (2014)
33. McGeoch, C.C., Wang, C.: Experimental evaluation of an adiabiatic quantum system for combinatorial optimization. In: Proc. of the ACM Intl. Conf. on Computing Frontiers. ACM (2013)
34. Mézard, M., Zecchina, R.: Random k-satisfiability problem: From an analytic solution to an efficient algorithm. Physical Review E **66**(5) (2002)
35. Monasson, R., Zecchina, R.: Entropy of the k-satisfiability problem. Physical review letters **76**(21) (1996)
36. Monasson, R., Zecchina, R., Kirkpatrick, S., Selman, B., Troyansky, L.: Determining computational complexity from characteristic phase transitions. Nature **400**(6740) (1999)
37. Morimae, T., Nishimura, H.: Merlinization of complexity classes above BQP. arXiv:1704.01514 (2017)
38. Moylett, D.J., Linden, N., Montanaro, A.: Quantum speedup of the traveling-salesman problem for bounded-degree graphs. Physical Review A **95**(3) (2017)
39. Murty, K.G., Kabadi, S.N.: Some NP-complete problems in quadratic and nonlinear programming. Mathematical programming **39**(2) (1987)
40. Neukart, F., Compostella, G., Seidel, C., von Dollen, D., Yarkoni, S., Parney, B.: Traffic flow optimization using a quantum annealer. Frontiers in ICT **4**, 29 (2017)
41. Strand, J., Przybysz, A., Ferguson, D., Zick, K.: ZZZ coupler for native embedding of MAX-3SAT problem instances in quantum annealing hardware. In: APS Meeting Abstracts (2017)
42. Van Dam, W., Mosca, M., Vazirani, U.: How powerful is adiabatic quantum computation? In: 42nd IEEE Symposium on Found. of Computer Science. IEEE (2001)
43. Warren, R.H.: Small traveling salesman problems. Journal of Advances in Applied Mathematics **2**(2) (2017)

Insights on Training Neural Networks for QUBO Tasks

Thomas Gabor, Sebastian Feld, Hila Safi, Thomy Phan, Claudia Linnhoff-Popien
thomas.gabor@ifi.lmu.de
LMU Munich

ABSTRACT

Current hardware limitations restrict the potential when solving quadratic unconstrained binary optimization (QUBO) problems via the quantum approximate optimization algorithm (QAOA) or quantum annealing (QA). Thus, we consider training neural networks in this context. We first discuss QUBO problems that originate from translated instances of the traveling salesman problem (TSP): Analyzing this representation via autoencoders shows that there is way more information included than necessary to solve the original TSP. Then we show that neural networks can be used to solve TSP instances from both QUBO input and autoencoders' hidden state representation. We finally generalize the approach and successfully train neural networks to solve arbitrary QUBO problems, sketching means to use neuromorphic hardware as a simulator or an additional co-processor for quantum computing.

CCS CONCEPTS

• Hardware → Quantum computation; **• Computing methodologies →** *Artificial intelligence*; **Neural networks**; **• Theory of computation →** Problems, reductions and completeness.

KEYWORDS

QUBO, quantum annealing, neural network, autoencoder

ACM Reference Format:
Thomas Gabor, Sebastian Feld, Hila Safi, Thomy Phan, Claudia Linnhoff-Popien. 2020. Insights on Training Neural Networks for QUBO Tasks. In *IEEE/ACM 42nd International Conference on Software Engineering Workshops (ICSEW'20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3387940.3391470

1 INTRODUCTION

Quadratic unconstrained binary optimization (QUBO) is a standard model for optimization problems (not only) in the quantum world as it can be used as input for algorithms like the quantum approximate optimization algorithm (QAOA) [4] or quantum annealing (QA) [9]. A QUBO instance of size n is given as an $n \times n$ matrix Q with $Q_{ij} \in \mathbb{R}$ for all $i, j \in \{1, ..., n\} \subseteq \mathbb{N}$. A solution to a QUBO instance Q is a vector $x^* \in \{0, 1\}^n$ so that

$$x^* = \arg\min_x \sum_{i \leq j} Q_{ij} x_i x_j.$$

Note that QUBO instances can trivially be derived from instances of Ising spin glasses [14]. Translations to QUBO and/or Ising models exist for a multitude of common optimization problems [6, 12], including many important NP-hard problems like 3-SAT [3] or scheduling problems [16].

In this paper, we focus on the well-known Traveling Salesman Problem (TSP): A TSP instance for m cities is given as an $m \times m$ matrix D with $D_{kl} \in \mathbb{R} \cup \{+\infty\}$ for all $k, l \in \mathbb{N}, 1 \leq k \leq m, 1 \leq l \leq m$. A solution to a TSP instance D is a vector $p^* \in \mathbb{N}^m$ that is a permutation of $(1, ..., m)$ and fulfills

$$p^* = \arg\min_p \; D_{p_m p_1} + \sum_{k=1}^{m-1} D_{p_k p_{k+1}}.$$

Despite apparent parallels in the formulation of a QUBO and TSP instances, the best translation from a TSP instance D for m cities produces a QUBO instance Q of size $n = m^2$, resulting roughly in a $m^2 \times m^2$ QUBO matrix with m^4 matrix cells in total [5]. This boost in size makes the QUBO translation rather inefficient for many practical applications and sometimes prohibits the resulting QUBO instances from being solved using quantum hardware at all, since current machines running QAOA or QA are severely limited in the amount of available qubits. However, since the computed QUBO instances originate from the smaller TSP instances, they clearly contain some redundant information.

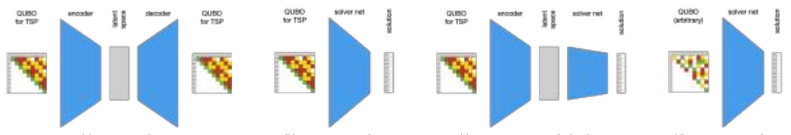
In order to assess alternative approaches to using the limited quantum hardware for solving QUBO problems, we apply neural networks (NNs) in this paper. These can help to bridge the gap until sufficiently large quantum hardware becomes available, but also may provide hooks for additional analysis. From a black-box perspective, a NN solving QUBOs can be treated like quantum annealer by the calling modules. Having such a mockup helps to identify which aspects of software engineering are really quantum-specific solution and which originate from the problem definition.

We explain the considered variants of NNs and how to apply them to work with problems formulated as QUBOs along the way. Using these NNs, we provide first empirical evidence for the following four hypotheses:

(1) Autoencoding QUBO instances generated from TSP instances is possible resulting in a hidden space having the size of the original TSP encoding (Fig. 1a, Sec. 2).
(2) NNs can be trained to solve QUBO instances generated from TSP instances (Fig 1b, Sec. 3).
(3) NNs can be trained to solve the encoded hidden spaces of these QUBO instances (Fig 1c, Sec. 4).
(4) NNs can be trained to solve arbitrary QUBO instances (Fig. 1d, Sec. 5).

An overview over the tested network architectures and setups is given in Figure 1. We discuss the lessons learned from these experiments and motivate further research in Sec. 6.

Taken from original publication: Thomas Gabor, Sebastian Feld, Hila Safi, Thomy Phan, and Claudia Linnhoff-Popien. Insights on training neural networks for QUBO tasks. In *First International Workshop on Quantum Software Engineering (Q-SE 2020)*, 2020



(a) Autoencoder **(b) TSP QUBO solver** **(c) TSP QUBO encoded solver** **(d) Any-QUBO solver**

Figure 1: Setups for testing the hypotheses.

2 AUTOENCODING QUBO FORMULATIONS OF TSP

Autoencoders (AEs) are NNs that typically possess an hourglass form: In the center they feature a hidden layer that is substantially smaller than the same-sized input and output layer. An AE is trained to reproduce its input data, but as the hidden layer is smaller than the input samples, they cannot simply "pass through" their inputs. Instead, the AE's first half (called *encoder*) needs to learn to abstract the most relevant features so that it can populate the *latent space*. This is the space of information that can be contained in the smallest hidden layer as densely as possible. Then, the second half (called *decoder*) will use this representation to reconstruct the original input as closely as possible. [7]

Once trained, AEs can be used to compress and decompress information by using the encoder and decoder part separately or to detect anomalies (i.e., input data not fitting the previously constructed latent space is assumed to substantially differ from previous training data). In our case, we use the process of training various AEs to estimate the entropy contained within the input data: the smallest latent space that still allows for almost no loss in autoencoding gives an estimate of the contained entropy in the data set, given that the encoder and decoder have been trained perfectly (if they have not, the estimate becomes rougher).

2.1 Setup

We have trained, tested and validated the network using different data sets. The training data consists of 11,000 randomly generated TSP instances that have been translated to QUBO; the test and validation data sets each consist of 1,000 samples.

There are different types of AEs, each with different advantages and disadvantages. The *vanilla autoencoder* represents the simplest form and consists of a network with three layers. After the input layer, a dense layer with a ReLU activation function reduces the input's dimensionality, followed by a second dense layer using sigmoid as an activation function that reconstructs the input. The *multilayer autoencoder* extends the previously described version by two more layers in both the encoder and decoder part. All layers use the ReLU activation function except the last layer, where the sigmoid activation function is used again. Finally, the *convolutional autoencoder* uses three-dimensional vectors instead of one-dimensional vectors, which is designed to be more suitable for compressing images and tested here for compressing matrices. Our setup consists of eleven layers: starting with an input layer, there are four encoder layers, two of which are pooling layers and the other two are convolutional layers with ReLU activation function. The decoder part consists of six layers: three convolutional layers with ReLU activation function, two upsampling layers and finally an output layer that uses the sigmoid activation function.

The initial layer set for each of the AEs is inspired by [7]. Depending on the type of layer (convolutional or dense), the input data's form must be adjusted. For the convolutional layer, a QUBO matrix is represented by an array of arrays. For the dense layer, the arrays have to be flattened, so QUBO problems are represented as a one-dimensional array in order to enable the network to recognize different problems. The final settings for each network were determined using various experiments and evaluations, which are presented in the following subsection.

The mean squared error (MSE) was used as a loss function for each AE since it shows a higher sensitivity to outliers than, for example, the absolute error. MSE calculates the average of the squared errors between predicted and actual output vectors.

The optimizers adam and stochastic gradient descent (SGD) were used to optimize the AE networks. Compared to SGD, adam, which was specially developed for training NNs, has the advantage that its learning rates are adaptive and potentially specific for each parameter. While adam uses little memory and converges faster, SGD is usually better at generalizing [11].

We measured accuracy using two methods: The *default accuracy* compares each predicted output with the actual output and returns the percentage of correctly predicted outputs. This process is repeated after each episode, with one episode corresponding to a training session on the entire input data set. However, this accuracy is of limited interest for our motivation, since we are rather interested in whether the shortest path is returned after encoding and decoding the QUBO matrix. Therefore, the *after-evaluation accuracy* was also used for training, test and evaluation. This consideration is necessary because there are at least two shortest tours in an undirected graph as for each tour there exist an opposite tour of the same length. Thus, the second accuracy uses the energy values of the solved QUBO problems, both regarding the actual qubit configuration and with the predicted ones. Accuracy is then calculated from the relationship between corresponding and all energies.

Each AE was trained for 600 epochs. Various learning rates were tries for the SGD optimizer, starting with a learning rate of 0.0001, 5,000 decay steps and a decay rate of 0.96. There were two further training setups with an initial training rate of 0.001 and 0.01, respectively [13]. For each network type, the best results were achieved using a learning rate of 0.001. Adam optimizer was

Taken from original publication: Thomas Gabor, Sebastian Feld, Hila Safi, Thomy Phan, and Claudia Linnhoff-Popien. Insights on training neural networks for QUBO tasks. In *First International Workshop on Quantum Software Engineering (Q-SE 2020)*, 2020

configured with no initial learning rate and in the event of poor optimization, the mentioned configurations for learning rate and decay were set. Batch size was set to 128. It turned out the AE using adam optimizer showed better results than the one using SGD.

2.2 Evaluation

Evaluating the AEs should identify whether (and to what extent) the QUBO representation of TSP instances can be reduced while at the same time being able to reconstruct the input. For this purpose, the NNs were trained and evaluated differently, starting with no reduction in dimensionality to a reduction of one fourth of the original size. The experiments started with TSP instances with 4 cities (4-TSP), i.e., a 16×16-sized QUBO matrix. Even though this problem size is not challenging for computers or humans, it served as a baseline for determining the best solution.

The vanilla AE has reconstructed the QUBO well up to a size of 50%. After that, the (after-evaluation) accuracy was below 40%. The accuracy of the multilayered autoencoder (MLAE) and the convolutional autoencoder (CAE) was at least 90%, even with a reduction to a quarter of the original size. For this reason, the vanilla AE was not evaluated further.

When encoding TSP instances with 8 cities (8-TSP), both AEs performed well; the CAE was slightly better. The after-evaluation accuracy of the MLAE is 0.95 for 4-TSP and 0.92 for 8-TSP. The CAE achieves an accuracy of 0.98 (4-TSP) and 0.95 (8-TSP). The default accuracy was 0.85 (MLAE) and 0.875 (CAE). The average energy difference of predictions that did not correspond to the actual energy was 5.2 for MLAE and 2.0 for CAE. Since CAE was best able to reconstruct the input, MLAE will not be further evaluated.

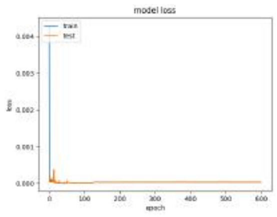
As CAE in combination with adam as the optimization function achieved the best results, this setup was chosen for the following experiments involving an encoder part.

In summary, it can be said that it is indeed possible to reduce the dimensionality of TSP instances represented as QUBO problems. A reduction to a size of one fourth shows that the QUBO matrices contain lots of redundant information. If a network for outputting the correct qubit configuration can be trained just using reduced input, training time can be drastically reduced. Fig. 2a and Fig. 2b show that the reduction task is quite simple for the AEs, since training converges already in early epochs.

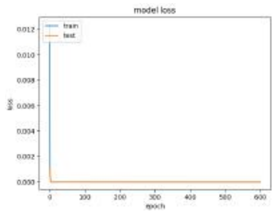
3 SOLVING QUBO FORMULATIONS OF TSP

The next step is to check whether a NN can be trained to solve a given QUBO problem. More specifically: is it possible to learn a qubit configuration that optimally solves a given problem.

The networks were again trained with a QUBO representation of TSP instances. However, since the required output differs from that of the AE part, new output data had to be generated accordingly. The required output for the NN is the qubit configuration for the shortest tour within the TSP instance. Corresponding qubit configurations were determined using qbsolv, a tool for operating the quantum annealing hardware by D-Wave Systems [8]. Qbsolv can also be used as a classical solver for QUBO problems.The functionality of qbsolv regarding the solution of TSP instances up to a size of 17 cities was checked and verified by comparing the tours returned



(a) Multilayer



(b) Convolutional

Figure 2: Autoencoder model loss for 4-TSP.

with those calculated using Google's OR-Tools [8] as well as with the solutions of the data sets by [2].

In order to determine a suitable NN for solving TSP instances, a recurrent neural network (RNN) and a convolutional neural network (CNN) were implemented. The results of both networks were compared, whereby again all networks were trained with a data set of size 11,000, and 1,000 samples each were used for test and validation.

3.1 Recurrent Neural Network

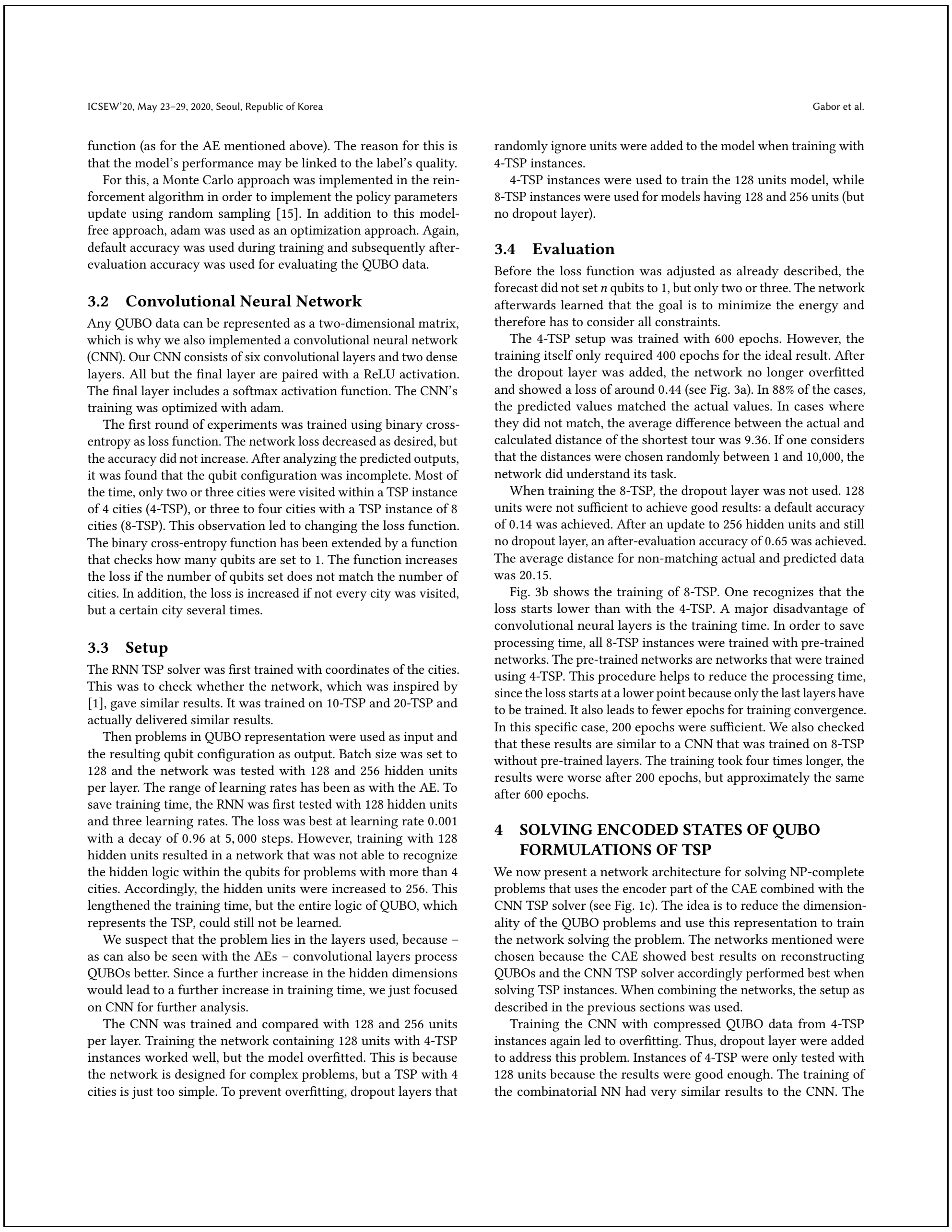
Our initial network model was inspired by [1]. They used one network architecture that solves both TSP and the likewise NP-complete knapsack problem. Their network uses the two-dimensional coordinates of the cities as input and the sequence of the cities to be visited as output. In our work, however, the input are TSP instances represented as QUBO matrices and the output is the shortest tour coded as a qubit configuration.

We use a pointer network consisting of two recurrent NN modules (encoder and decoder). As in [1], we implement attention using long short-term memory (LSTM) cells [10].

The loss is calculated using binary cross-entropy. This loss function is suitable for problems with a yes/no decision, which is the case with our 0/1 output representing the qubit configuration.

With regard to the optimizer function for training the RNN, we have strictly adhered to the structure of [1]. They propose to use optimization via policy gradients instead of a supervised loss

Taken from original publication: Thomas Gabor, Sebastian Feld, Hila Safi, Thomy Phan, and Claudia Linnhoff-Popien. Insights on training neural networks for QUBO tasks. In *First International Workshop on Quantum Software Engineering (Q-SE 2020)*, 2020



function (as for the AE mentioned above). The reason for this is that the model's performance may be linked to the label's quality.

For this, a Monte Carlo approach was implemented in the reinforcement algorithm in order to implement the policy parameters update using random sampling [15]. In addition to this model-free approach, adam was used as an optimization approach. Again, default accuracy was used during training and subsequently after-evaluation accuracy was used for evaluating the QUBO data.

3.2 Convolutional Neural Network

Any QUBO data can be represented as a two-dimensional matrix, which is why we also implemented a convolutional neural network (CNN). Our CNN consists of six convolutional layers and two dense layers. All but the final layer are paired with a ReLU activation. The final layer includes a softmax activation function. The CNN's training was optimized with adam.

The first round of experiments was trained using binary cross-entropy as loss function. The network loss decreased as desired, but the accuracy did not increase. After analyzing the predicted outputs, it was found that the qubit configuration was incomplete. Most of the time, only two or three cities were visited within a TSP instance of 4 cities (4-TSP), or three to four cities with a TSP instance of 8 cities (8-TSP). This observation led to changing the loss function. The binary cross-entropy function has been extended by a function that checks how many qubits are set to 1. The function increases the loss if the number of qubits set does not match the number of cities. In addition, the loss is increased if not every city was visited, but a certain city several times.

3.3 Setup

The RNN TSP solver was first trained with coordinates of the cities. This was to check whether the network, which was inspired by [1], gave similar results. It was trained on 10-TSP and 20-TSP and actually delivered similar results.

Then problems in QUBO representation were used as input and the resulting qubit configuration as output. Batch size was set to 128 and the network was tested with 128 and 256 hidden units per layer. The range of learning rates has been as with the AE. To save training time, the RNN was first tested with 128 hidden units and three learning rates. The loss was best at learning rate 0.001 with a decay of 0.96 at $5,000$ steps. However, training with 128 hidden units resulted in a network that was not able to recognize the hidden logic within the qubits for problems with more than 4 cities. Accordingly, the hidden units were increased to 256. This lengthened the training time, but the entire logic of QUBO, which represents the TSP, could still not be learned.

We suspect that the problem lies in the layers used, because – as can also be seen with the AEs – convolutional layers process QUBOs better. Since a further increase in the hidden dimensions would lead to a further increase in training time, we just focused on CNN for further analysis.

The CNN was trained and compared with 128 and 256 units per layer. Training the network containing 128 units with 4-TSP instances worked well, but the model overfitted. This is because the network is designed for complex problems, but a TSP with 4 cities is just too simple. To prevent overfitting, dropout layers that randomly ignore units were added to the model when training with 4-TSP instances.

4-TSP instances were used to train the 128 units model, while 8-TSP instances were used for models having 128 and 256 units (but no dropout layer).

3.4 Evaluation

Before the loss function was adjusted as already described, the forecast did not set n qubits to 1, but only two or three. The network afterwards learned that the goal is to minimize the energy and therefore has to consider all constraints.

The 4-TSP setup was trained with 600 epochs. However, the training itself only required 400 epochs for the ideal result. After the dropout layer was added, the network no longer overfitted and showed a loss of around 0.44 (see Fig. 3a). In 88% of the cases, the predicted values matched the actual values. In cases where they did not match, the average difference between the actual and calculated distance of the shortest tour was 9.36. If one considers that the distances were chosen randomly between 1 and 10,000, the network did understand its task.

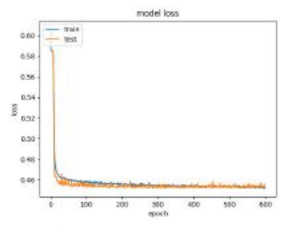
When training the 8-TSP, the dropout layer was not used. 128 units were not sufficient to achieve good results: a default accuracy of 0.14 was achieved. After an update to 256 hidden units and still no dropout layer, an after-evaluation accuracy of 0.65 was achieved. The average distance for non-matching actual and predicted data was 20.15.

Fig. 3b shows the training of 8-TSP. One recognizes that the loss starts lower than with the 4-TSP. A major disadvantage of convolutional neural layers is the training time. In order to save processing time, all 8-TSP instances were trained with pre-trained networks. The pre-trained networks are networks that were trained using 4-TSP. This procedure helps to reduce the processing time, since the loss starts at a lower point because only the last layers have to be trained. It also leads to fewer epochs for training convergence. In this specific case, 200 epochs were sufficient. We also checked that these results are similar to a CNN that was trained on 8-TSP without pre-trained layers. The training took four times longer, the results were worse after 200 epochs, but approximately the same after 600 epochs.

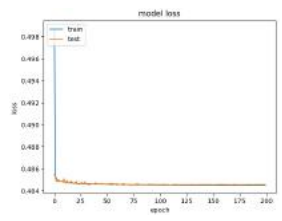
4 SOLVING ENCODED STATES OF QUBO FORMULATIONS OF TSP

We now present a network architecture for solving NP-complete problems that uses the encoder part of the CAE combined with the CNN TSP solver (see Fig. 1c). The idea is to reduce the dimensionality of the QUBO problems and use this representation to train the network solving the problem. The networks mentioned were chosen because the CAE showed best results on reconstructing QUBOs and the CNN TSP solver accordingly performed best when solving TSP instances. When combining the networks, the setup as described in the previous sections was used.

Training the CNN with compressed QUBO data from 4-TSP instances again led to overfitting. Thus, dropout layer were added to address this problem. Instances of 4-TSP were only tested with 128 units because the results were good enough. The training of the combinatorial NN had very similar results to the CNN. The



(a) 4**-TSP,** 128 **units, dropout layer**



(b) 8**-TSP,** 256 **units, pre-trained**

Figure 3: Convolutional neural network model loss.

loss converged at 0.46 and had a default accuracy of 0.75 and an after-evaluation accuracy of 0.85. The average difference between all non-matching and actual results was 9.56. The network was trained over 600 epochs.

The compression of the input had almost no effect on the network's ability to learn qubit configurations. The network's training time was highly reduced when only compressed input was used. The CNN used about $9 - 10$ hours of training time, while the combinatorial NN only used about $4 - 5$ hours.

In order to learn 8-TSP instances, the combinatorial NN has again used pre-trained layers, i.e., those of the 4-TSP combinatorial NN. Again, there are only minor differences from the CNN results. The loss was 0.48 (see Fig. 4), which is identical to the CNN's loss. The network was trained for 200 epochs, had a default accuracy of 0.55, an after-evaluation accuracy of 0.64, and a mean difference between non-matching and actual results of 27.5. This value is 7.35 higher than that of the CNN, but still acceptable as the cities' distances were randomly chosen between 1 and 10,000.

5 SOLVING ARBITRARY QUBO INSTANCES

Finally, we want to take another step towards generalization and train NNs to solve arbitrary QUBOs. In this way, they can be functionally used in place of a quantum annealing solver.

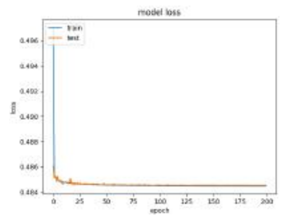


Figure 4: Combinatorial NN model loss, 8**-TSP,** 256 **units, pre-trained.**

5.1 Setup

Random QUBOs have no inherent structure that could be exploited by an AE, so we only trained CNNs for this task. The input data was generated by filling the upper triangular matrix with random numbers between −10,000 and 10,000. The output was generated by labeling given input with the qubit configurations that were created using qbsolv. The training data set consisted of 11,000 samples, the validation and test data set each consisted of 1,000 samples.

5.2 Evaluation

We want to show that a single NN can solve not only a specific NP-complete problem, but a generic one. In order to obtain comparability, random QUBOs were created that have the same dimensionality as 4-TSP and 8-TSP QUBOs.

The CNN was able to learn from the random QUBOs: 16×16-dimensional matrices (equivalent to 4-TSP) were trained using 256 units per layer over 400 epochs and had a loss of 1.06. The default accuracy was 0.45, the after-evaluation accuracy 0.48. The mean energy difference between non-matching actual and predicted results was 230.32, which is much higher than with TSP. However, the qubit configuration was correct for almost every second result.

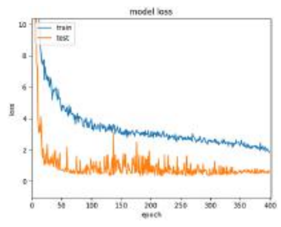
Training using random data is far more complex than training a specific problem (see Fig. 5a and Fig. 5b). The network did not overfit, not even with twice as many units. The random 64×64-sized QUBO problems (equivalent to 8-TSP) were trained with a network having 256 units per layer over 1, 800 epochs and had a loss of 9.05. Default accuracy and after-evaluation accuracy were around 0.2 and the average energy difference of the non-matching outputs was 345.45.

Training with random values took a lot of time for relevant QUBO sizes, the accuracy fell faster with the increase in the QUBOs' dimensionality than with TSP. The use of a network pre-trained on 16×16-sized random QUBO problems inside a 64×64 random QUBO network had an accuracy of 0.12 and did not work comparable to the CNN. In addition to the fact that an energy minimum is sought, the larger network cannot reuse much information.

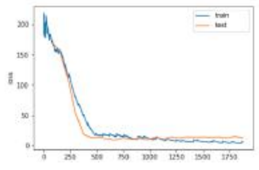
6 CONCLUSION

We provided empirical evidence for four hypotheses. (1) AEs are able to filter the overhead induced by a QUBO translation of TSP to

Taken from original publication: Thomas Gabor, Sebastian Feld, Hila Safi, Thomy Phan, and Claudia Linnhoff-Popien. Insights on training neural networks for QUBO tasks. In *First International Workshop on Quantum Software Engineering (Q-SE 2020)*, 2020



(a) 16 × 16



(b) 64 × 64

Figure 5: Random QUBO Solver model loss.

some extent. They can thus be used to guess the original complexity of a problem from its QUBO formulation. (2) NNs can be trained to return the qubit configuration resulting in minimum energy for a QUBO problem generated from a TSP instance. They are thus able to solve TSP even in a larger QUBO translation. (3) Accordingly, NNs can also solve QUBO problems originating from TSP given their latent space representation (instead of the full QUBO matrix). (4) NNs can be trained to solve QUBO problems in general. The fact that CNNs appear most effective implies that QUBO problems can be treated more like a somewhat local graph problem and less like combinatorial optimization.

These first steps call for immediate follow-up research. Most importantly, a thorough study of the various impact of overhead from the QUBO translation is necessary: How do networks that have been trained for (a) solving TSP in native encoding, (b) solving QUBO translations of TSP, and (c) solving QUBO in general compare on the same set of problems regarding various performance metrics? Are there cases where a QUBO translation may actually be easier to solve than other representations of TSP? Does specialized training on just one type of QUBO bring any advantage over training on random QUBOs? How do the results on TSP (whose QUBO translation introduces a quadratic overhead) compare to problems with more (or less) efficient QUBO translations?

From this experience report, a strong argument can be made for mathematically solid interfaces in quantum computing: The NNs we trained should be able to replace any other means of solving QUBOs fully transparent to the provider of the problem instances. A diverse pool of mechanisms for solving QUBOs should prove useful to establish QUBO as a suitable formulation for optimization problems and thus prepare for the eventual deployment of quantum-based machines. Current breakthrough technology like neuromorphic hardware may thus serve as a bridge to the quantum age.

We argue that for some time to come, quantum software will usually only be shipped as a module within larger, mostly classical software applications. Furthermore, these modules will usually come with fully classical counterparts as quantum resources will remain comparatively limited and thus should not be used up unnecessarily, for example when testing other parts of the software where a good enough approximation of the quantum module suffices. We think that NNs may provide a very generic tool to produce such counterparts as it has been done in this case study for quantum annealing or QAOA, even though their rather black-box nature opens up a new field of testing issues. Effectively, we argue that any approach to the integration of quantum modules should aim to include similar classical approximation models at least for the near future.

We would like to point out that even in the presence of large-scale quantum hardware, handling QUBO problems with NNs might still be useful for pre- and post-processing of problem instances, dispatching instances to various hardware platforms, or providing estimates of the inherent complexity of a specific problem or problem instance. As we have shown that NNs can handle the structure of QUBO matrices well, they may also be able to learn transformations (ideally with automatic reduction of size) on them or help with introspection of the optimization process and effectively the debugging of optimization problem formulations or quantum hardware platforms.

REFERENCES

[1] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. 2016. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940* (2016).

[2] John Burkardt. 2019. *TSP – Data for the Traveling Salesperson Problem.* people.sc.fsu.edu/~jburkardt/datasets/tsp/tsp.html

[3] Vicky Choi. 2010. Adiabatic quantum algorithms for the NP-complete Maximum-Weight Independent set, Exact Cover and 3SAT problems. *arXiv preprint arXiv:1004.2226* (2010).

[4] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. 2014. A quantum approximate optimization algorithm. *arXiv preprint arXiv:1411.4028* (2014).

[5] Sebastian Feld, Christoph Roch, Thomas Gabor, Christian Seidel, Florian Neukart, Isabella Galter, Wolfgang Mauerer, and Claudia Linnhoff-Popien. 2018. A hybrid solution method for the capacitated vehicle routing problem using a quantum annealer. *arXiv preprint arXiv:1811.07403* (2018).

[6] Fred Glover, Gary Kochenberger, and Yu Du. 2018. A tutorial on formulating and using QUBO models. *arXiv preprint arXiv:1811.11538* (2018).

[7] Nathan Hubens. 2018. *Deep inside: Autoencoders.* tinyurl.com/ucu9ump

[8] D-Wave Systems Inc. 2019. *qbsolv – qbsolv documentation.* docs.ocean.dwavesys.com/projects/qbsolv/en/latest/

[9] Tadashi Kadowaki and Hidetoshi Nishimori. 1998. Quantum annealing in the transverse Ising model. *Physical Review E* 58, 5 (1998), 5355.

[10] Anusha Lihala. 2019. *Attention and its Different Forms.* towardsdatascience.com/attention-and-its-different-forms-7fc3674d14dc

[11] Shao-Anb Lu. 2017. *SGD > Adam?? Which One Is The Best Optimizer: Dogs-VS-Cats Toy Experiment.* tinyurl.com/sanjtd4

[12] Andrew Lucas. 2014. Ising formulations of many NP problems. *Frontiers in Physics* 2 (2014), 5.

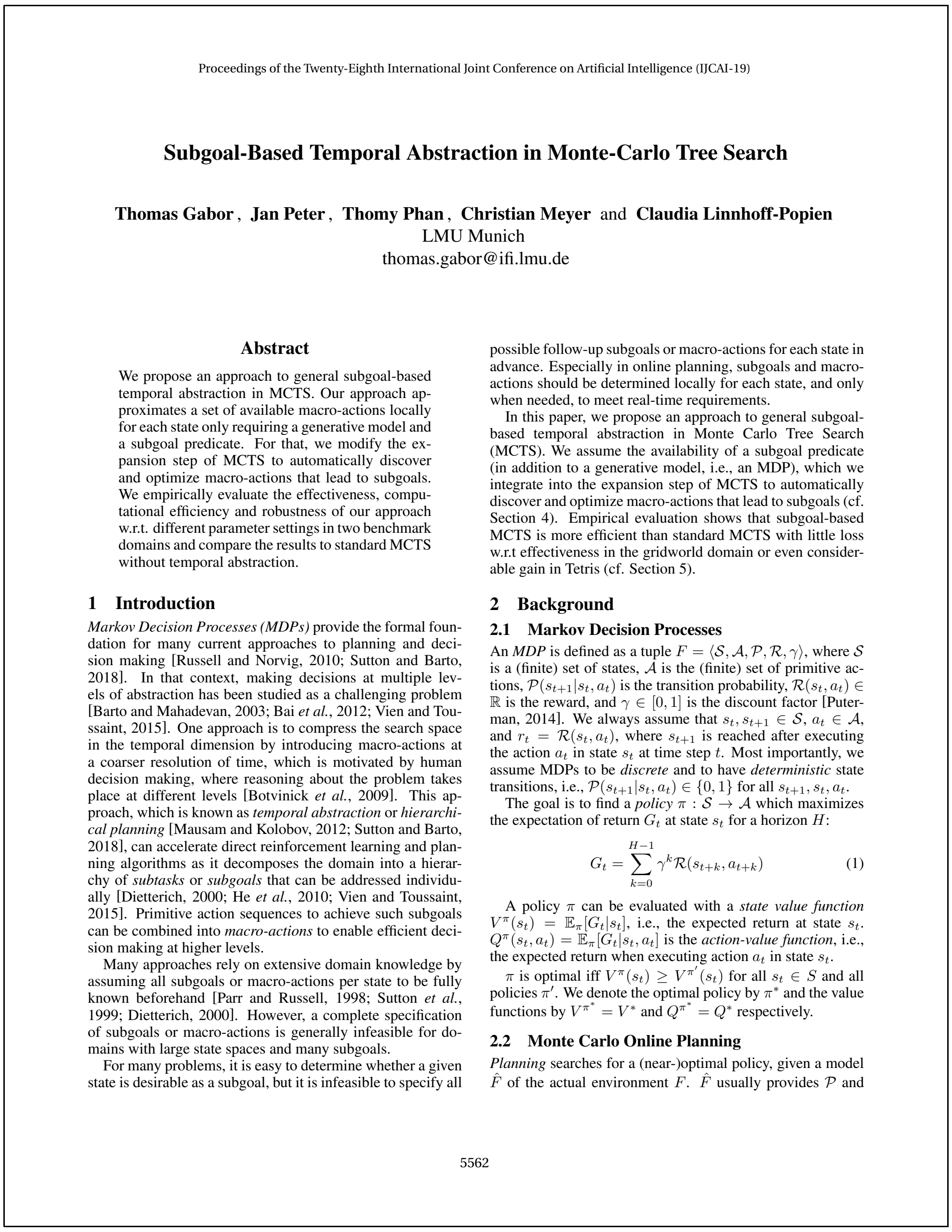
[13] David Mack. 2018. *How to pick the best learning rate for your machine learning project.* tinyurl.com/rtnglcu

[14] Catherine C McGeoch. 2014. Adiabatic quantum computation and quantum annealing: Theory and practice. *Synthesis Lectures on QC* 5, 2 (2014), 1–93.

[15] Madhu Sanjeevi. 2018. *Model Free Reinforcement learning algorithms (Monte Carlo, SARSA, Q-learning).* tinyurl.com/rooumla

[16] Tobias Stollenwerk and Achim Basermann. 2016. Experiences with scheduling problems on adiabatic quantum computers. In *1st Int'l Workshop on Post-Moore Era Supercomputing (PMES).* Future Technologies Group, 45–46.

Taken from original publication: Thomas Gabor, Jan Peter, Thomy Phan, Christian Meyer, and Claudia Linnhoff-Popien. Subgoal-based temporal abstraction in Monte-Carlo tree search. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 5562–5568. AAAI Press, 2019



Subgoal-Based Temporal Abstraction in Monte-Carlo Tree Search

Thomas Gabor , **Jan Peter** , **Thomy Phan** , **Christian Meyer** and **Claudia Linnhoff-Popien**
LMU Munich
thomas.gabor@ifi.lmu.de

Abstract

We propose an approach to general subgoal-based temporal abstraction in MCTS. Our approach approximates a set of available macro-actions locally for each state only requiring a generative model and a subgoal predicate. For that, we modify the expansion step of MCTS to automatically discover and optimize macro-actions that lead to subgoals. We empirically evaluate the effectiveness, computational efficiency and robustness of our approach w.r.t. different parameter settings in two benchmark domains and compare the results to standard MCTS without temporal abstraction.

1 Introduction

Markov Decision Processes (MDPs) provide the formal foundation for many current approaches to planning and decision making [Russell and Norvig, 2010; Sutton and Barto, 2018]. In that context, making decisions at multiple levels of abstraction has been studied as a challenging problem [Barto and Mahadevan, 2003; Bai *et al.*, 2012; Vien and Toussaint, 2015]. One approach is to compress the search space in the temporal dimension by introducing macro-actions at a coarser resolution of time, which is motivated by human decision making, where reasoning about the problem takes place at different levels [Botvinick *et al.*, 2009]. This approach, which is known as *temporal abstraction* or *hierarchical planning* [Mausam and Kolobov, 2012; Sutton and Barto, 2018], can accelerate direct reinforcement learning and planning algorithms as it decomposes the domain into a hierarchy of *subtasks* or *subgoals* that can be addressed individually [Dietterich, 2000; He *et al.*, 2010; Vien and Toussaint, 2015]. Primitive action sequences to achieve such subgoals can be combined into *macro-actions* to enable efficient decision making at higher levels.

Many approaches rely on extensive domain knowledge by assuming all subgoals or macro-actions per state to be fully known beforehand [Parr and Russell, 1998; Sutton *et al.*, 1999; Dietterich, 2000]. However, a complete specification of subgoals or macro-actions is generally infeasible for domains with large state spaces and many subgoals.

For many problems, it is easy to determine whether a given state is desirable as a subgoal, but it is infeasible to specify all possible follow-up subgoals or macro-actions for each state in advance. Especially in online planning, subgoals and macro-actions should be determined locally for each state, and only when needed, to meet real-time requirements.

In this paper, we propose an approach to general subgoal-based temporal abstraction in Monte Carlo Tree Search (MCTS). We assume the availability of a subgoal predicate (in addition to a generative model, i.e., an MDP), which we integrate into the expansion step of MCTS to automatically discover and optimize macro-actions that lead to subgoals (cf. Section 4). Empirical evaluation shows that subgoal-based MCTS is more efficient than standard MCTS with little loss w.r.t effectiveness in the gridworld domain or even considerable gain in Tetris (cf. Section 5).

2 Background

2.1 Markov Decision Processes

An *MDP* is defined as a tuple $F = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where $\mathcal{S}$ is a (finite) set of states, $\mathcal{A}$ is the (finite) set of primitive actions, $\mathcal{P}(s_{t+1}|s_t, a_t)$ is the transition probability, $\mathcal{R}(s_t, a_t) \in \mathbb{R}$ is the reward, and $\gamma \in [0, 1]$ is the discount factor [Puterman, 2014]. We always assume that $s_t, s_{t+1} \in \mathcal{S}$, $a_t \in \mathcal{A}$, and $r_t = \mathcal{R}(s_t, a_t)$, where s_{t+1} is reached after executing the action a_t in state s_t at time step t. Most importantly, we assume MDPs to be *discrete* and to have *deterministic* state transitions, i.e., $\mathcal{P}(s_{t+1}|s_t, a_t) \in \{0, 1\}$ for all s_{t+1}, s_t, a_t.

The goal is to find a *policy* $\pi : \mathcal{S} \to \mathcal{A}$ which maximizes the expectation of return G_t at state s_t for a horizon H:

$$G_t = \sum_{k=0}^{H-1} \gamma^k \mathcal{R}(s_{t+k}, a_{t+k}) \tag{1}$$

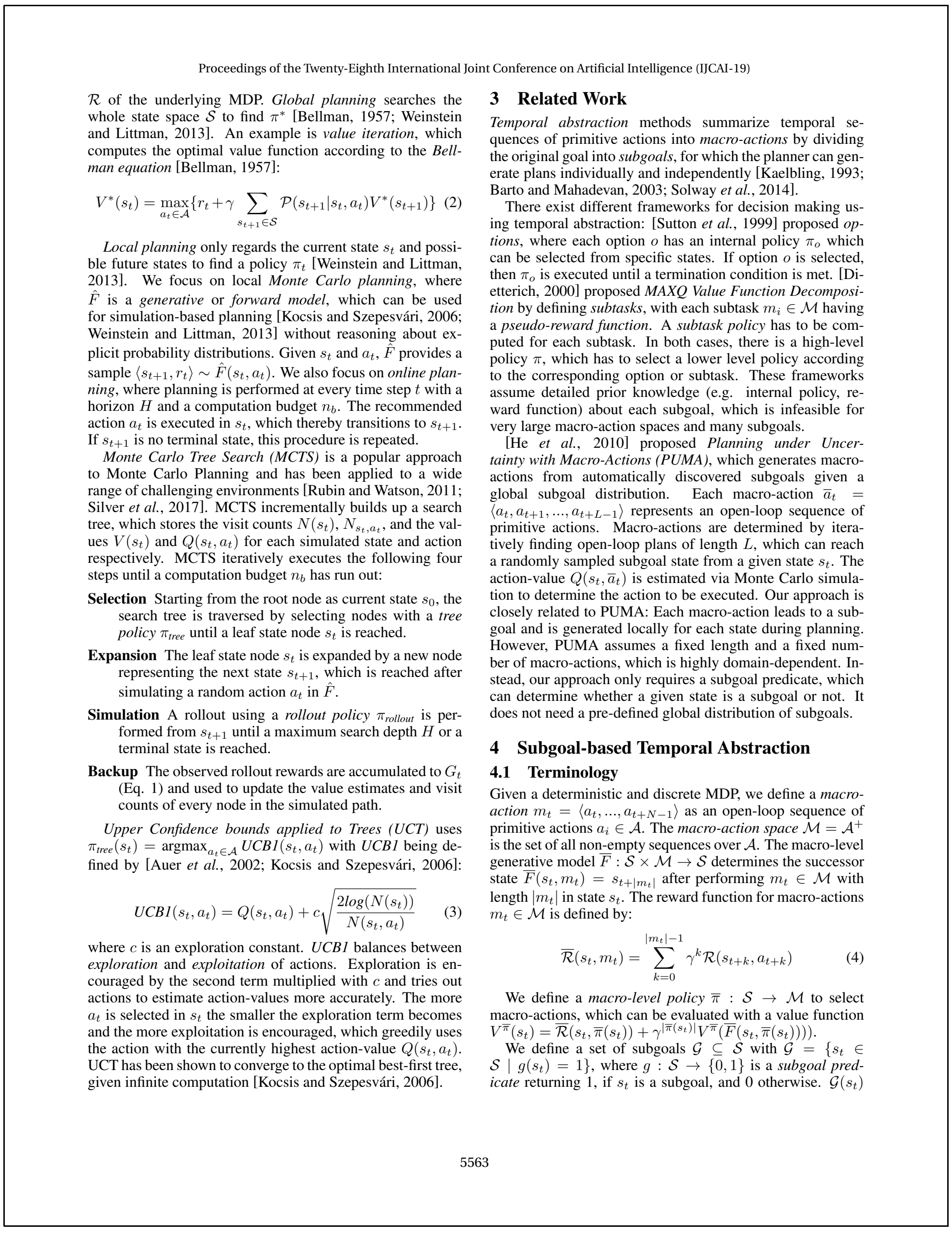
A policy π can be evaluated with a *state value function* $V^\pi(s_t) = \mathbb{E}_\pi[G_t|s_t]$, i.e., the expected return at state s_t. $Q^\pi(s_t, a_t) = \mathbb{E}_\pi[G_t|s_t, a_t]$ is the *action-value function*, i.e., the expected return when executing action a_t in state s_t.

π is optimal iff $V^\pi(s_t) \geq V^{\pi'}(s_t)$ for all $s_t \in S$ and all policies π'. We denote the optimal policy by π^* and the value functions by $V^{\pi^*} = V^*$ and $Q^{\pi^*} = Q^*$ respectively.

2.2 Monte Carlo Online Planning

Planning searches for a (near-)optimal policy, given a model $\hat{F}$ of the actual environment F. $\hat{F}$ usually provides $\mathcal{P}$ and

Taken from original publication: Thomas Gabor, Jan Peter, Thomy Phan, Christian Meyer, and Claudia Linnhoff-Popien. Subgoal-based temporal abstraction in Monte-Carlo tree search. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 5562–5568. AAAI Press, 2019



$\mathcal{R}$ of the underlying MDP. *Global planning* searches the whole state space $\mathcal{S}$ to find π^* [Bellman, 1957; Weinstein and Littman, 2013]. An example is *value iteration*, which computes the optimal value function according to the *Bellman equation* [Bellman, 1957]:

$$V^*(s_t) = \max_{a_t \in \mathcal{A}} \{r_t + \gamma \sum_{s_{t+1} \in \mathcal{S}} \mathcal{P}(s_{t+1}|s_t, a_t) V^*(s_{t+1})\} \quad (2)$$

Local planning only regards the current state s_t and possible future states to find a policy π_t [Weinstein and Littman, 2013]. We focus on local *Monte Carlo planning*, where $\hat{F}$ is a *generative* or *forward model*, which can be used for simulation-based planning [Kocsis and Szepesvári, 2006; Weinstein and Littman, 2013] without reasoning about explicit probability distributions. Given s_t and a_t, $\hat{F}$ provides a sample $\langle s_{t+1}, r_t \rangle \sim \hat{F}(s_t, a_t)$. We also focus on *online planning*, where planning is performed at every time step t with a horizon H and a computation budget n_b. The recommended action a_t is executed in s_t, which thereby transitions to s_{t+1}. If s_{t+1} is no terminal state, this procedure is repeated.

Monte Carlo Tree Search (MCTS) is a popular approach to Monte Carlo Planning and has been applied to a wide range of challenging environments [Rubin and Watson, 2011; Silver *et al.*, 2017]. MCTS incrementally builds up a search tree, which stores the visit counts $N(s_t)$, N_{s_t,a_t}, and the values $V(s_t)$ and $Q(s_t, a_t)$ for each simulated state and action respectively. MCTS iteratively executes the following four steps until a computation budget n_b has run out:

Selection Starting from the root node as current state s_0, the search tree is traversed by selecting nodes with a *tree policy* π_{tree} until a leaf state node s_t is reached.

Expansion The leaf state node s_t is expanded by a new node representing the next state s_{t+1}, which is reached after simulating a random action a_t in $\hat{F}$.

Simulation A rollout using a *rollout policy* $\pi_{rollout}$ is performed from s_{t+1} until a maximum search depth H or a terminal state is reached.

Backup The observed rollout rewards are accumulated to G_t (Eq. 1) and used to update the value estimates and visit counts of every node in the simulated path.

Upper Confidence bounds applied to Trees (UCT) uses $\pi_{tree}(s_t) = \text{argmax}_{a_t \in \mathcal{A}} \, UCB1(s_t, a_t)$ with *UCB1* being defined by [Auer *et al.*, 2002; Kocsis and Szepesvári, 2006]:

$$UCB1(s_t, a_t) = Q(s_t, a_t) + c\sqrt{\frac{2log(N(s_t))}{N(s_t, a_t)}} \quad (3)$$

where c is an exploration constant. *UCB1* balances between *exploration* and *exploitation* of actions. Exploration is encouraged by the second term multiplied with c and tries out actions to estimate action-values more accurately. The more a_t is selected in s_t the smaller the exploration term becomes and the more exploitation is encouraged, which greedily uses the action with the currently highest action-value $Q(s_t, a_t)$. UCT has been shown to converge to the optimal best-first tree, given infinite computation [Kocsis and Szepesvári, 2006].

3 Related Work

Temporal abstraction methods summarize temporal sequences of primitive actions into *macro-actions* by dividing the original goal into *subgoals*, for which the planner can generate plans individually and independently [Kaelbling, 1993; Barto and Mahadevan, 2003; Solway *et al.*, 2014].

There exist different frameworks for decision making using temporal abstraction: [Sutton *et al.*, 1999] proposed *options*, where each option o has an internal policy π_o which can be selected from specific states. If option o is selected, then π_o is executed until a termination condition is met. [Dietterich, 2000] proposed *MAXQ Value Function Decomposition* by defining *subtasks*, with each subtask $m_i \in \mathcal{M}$ having a *pseudo-reward function*. A *subtask policy* has to be computed for each subtask. In both cases, there is a high-level policy π, which has to select a lower level policy according to the corresponding option or subtask. These frameworks assume detailed prior knowledge (e.g. internal policy, reward function) about each subgoal, which is infeasible for very large macro-action spaces and many subgoals.

[He *et al.*, 2010] proposed *Planning under Uncertainty with Macro-Actions (PUMA)*, which generates macro-actions from automatically discovered subgoals given a global subgoal distribution. Each macro-action $\overline{a}_t = \langle a_t, a_{t+1}, ..., a_{t+L-1} \rangle$ represents an open-loop sequence of primitive actions. Macro-actions are determined by iteratively finding open-loop plans of length L, which can reach a randomly sampled subgoal state from a given state s_t. The action-value $Q(s_t, \overline{a}_t)$ is estimated via Monte Carlo simulation to determine the action to be executed. Our approach is closely related to PUMA: Each macro-action leads to a subgoal and is generated locally for each state during planning. However, PUMA assumes a fixed length and a fixed number of macro-actions, which is highly domain-dependent. Instead, our approach only requires a subgoal predicate, which can determine whether a given state is a subgoal or not. It does not need a pre-defined global distribution of subgoals.

4 Subgoal-based Temporal Abstraction

4.1 Terminology

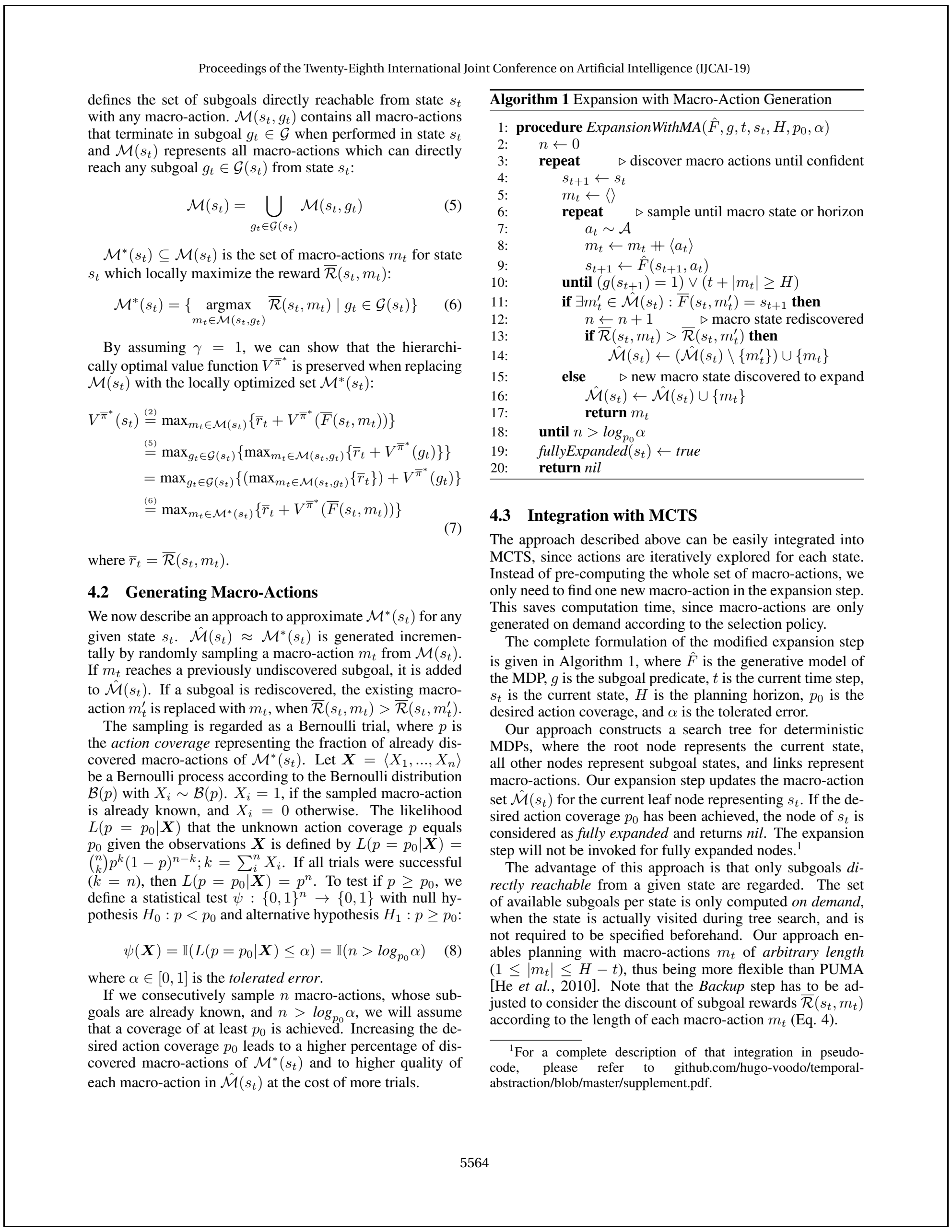
Given a deterministic and discrete MDP, we define a *macro-action* $m_t = \langle a_t, ..., a_{t+N-1} \rangle$ as an open-loop sequence of primitive actions $a_i \in \mathcal{A}$. The *macro-action space* $\mathcal{M} = \mathcal{A}^+$ is the set of all non-empty sequences over $\mathcal{A}$. The macro-level generative model $\overline{F} : \mathcal{S} \times \mathcal{M} \rightarrow \mathcal{S}$ determines the successor state $\overline{F}(s_t, m_t) = s_{t+|m_t|}$ after performing $m_t \in \mathcal{M}$ with length $|m_t|$ in state s_t. The reward function for macro-actions $m_t \in \mathcal{M}$ is defined by:

$$\overline{\mathcal{R}}(s_t, m_t) = \sum_{k=0}^{|m_t|-1} \gamma^k \mathcal{R}(s_{t+k}, a_{t+k}) \quad (4)$$

We define a *macro-level policy* $\overline{\pi} : \mathcal{S} \rightarrow \mathcal{M}$ to select macro-actions, which can be evaluated with a value function $V^{\overline{\pi}}(s_t) = \overline{\mathcal{R}}(s_t, \overline{\pi}(s_t)) + \gamma^{|\overline{\pi}(s_t)|} V^{\overline{\pi}}(\overline{F}(s_t, \overline{\pi}(s_t))))$.

We define a set of subgoals $\mathcal{G} \subseteq \mathcal{S}$ with $\mathcal{G} = \{s_t \in \mathcal{S} \mid g(s_t) = 1\}$, where $g : \mathcal{S} \rightarrow \{0, 1\}$ is a *subgoal predicate* returning 1, if s_t is a subgoal, and 0 otherwise. $\mathcal{G}(s_t)$

Taken from original publication: Thomas Gabor, Jan Peter, Thomy Phan, Christian Meyer, and Claudia Linnhoff-Popien. Subgoal-based temporal abstraction in Monte-Carlo tree search. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 5562–5568. AAAI Press, 2019



defines the set of subgoals directly reachable from state s_t with any macro-action. $\mathcal{M}(s_t, g_t)$ contains all macro-actions that terminate in subgoal $g_t \in \mathcal{G}$ when performed in state s_t and $\mathcal{M}(s_t)$ represents all macro-actions which can directly reach any subgoal $g_t \in \mathcal{G}(s_t)$ from state s_t:

$$\mathcal{M}(s_t) = \bigcup_{g_t \in \mathcal{G}(s_t)} \mathcal{M}(s_t, g_t) \tag{5}$$

$\mathcal{M}^*(s_t) \subseteq \mathcal{M}(s_t)$ is the set of macro-actions m_t for state s_t which locally maximize the reward $\overline{\mathcal{R}}(s_t, m_t)$:

$$\mathcal{M}^*(s_t) = \{ \underset{m_t \in \mathcal{M}(s_t, g_t)}{\text{argmax}} \ \overline{\mathcal{R}}(s_t, m_t) \mid g_t \in \mathcal{G}(s_t)\} \tag{6}$$

By assuming $\gamma = 1$, we can show that the hierarchically optimal value function $V^{\overline{\pi}^*}$ is preserved when replacing $\mathcal{M}(s_t)$ with the locally optimized set $\mathcal{M}^*(s_t)$:

$$\begin{aligned} V^{\overline{\pi}^*}(s_t) &\overset{(2)}{=} \max\nolimits_{m_t \in \mathcal{M}(s_t)} \{\overline{r}_t + V^{\overline{\pi}^*}(\overline{F}(s_t, m_t))\} \\ &\overset{(5)}{=} \max\nolimits_{g_t \in \mathcal{G}(s_t)} \{\max\nolimits_{m_t \in \mathcal{M}(s_t, g_t)} \{\overline{r}_t + V^{\overline{\pi}^*}(g_t)\}\} \\ &= \max\nolimits_{g_t \in \mathcal{G}(s_t)} \{(\max\nolimits_{m_t \in \mathcal{M}(s_t, g_t)} \{\overline{r}_t\}) + V^{\overline{\pi}^*}(g_t)\} \\ &\overset{(6)}{=} \max\nolimits_{m_t \in \mathcal{M}^*(s_t)} \{\overline{r}_t + V^{\overline{\pi}^*}(\overline{F}(s_t, m_t))\} \end{aligned} \tag{7}$$

where $\overline{r}_t = \overline{\mathcal{R}}(s_t, m_t)$.

4.2 Generating Macro-Actions

We now describe an approach to approximate $\mathcal{M}^*(s_t)$ for any given state s_t. $\hat{\mathcal{M}}(s_t) \approx \mathcal{M}^*(s_t)$ is generated incrementally by randomly sampling a macro-action m_t from $\mathcal{M}(s_t)$. If m_t reaches a previously undiscovered subgoal, it is added to $\hat{\mathcal{M}}(s_t)$. If a subgoal is rediscovered, the existing macro-action m'_t is replaced with m_t, when $\overline{\mathcal{R}}(s_t, m_t) > \overline{\mathcal{R}}(s_t, m'_t)$.

The sampling is regarded as a Bernoulli trial, where p is the *action coverage* representing the fraction of already discovered macro-actions of $\mathcal{M}^*(s_t)$. Let $\boldsymbol{X} = \langle X_1, ..., X_n \rangle$ be a Bernoulli process according to the Bernoulli distribution $\mathcal{B}(p)$ with $X_i \sim \mathcal{B}(p)$. $X_i = 1$, if the sampled macro-action is already known, and $X_i = 0$ otherwise. The likelihood $L(p = p_0 | \boldsymbol{X})$ that the unknown action coverage p equals p_0 given the observations $\boldsymbol{X}$ is defined by $L(p = p_0 | \boldsymbol{X}) = \binom{n}{k} p^k (1-p)^{n-k}; k = \sum_i^n X_i$. If all trials were successful ($k = n$), then $L(p = p_0 | \boldsymbol{X}) = p^n$. To test if $p \geq p_0$, we define a statistical test $\psi : \{0,1\}^n \rightarrow \{0,1\}$ with null hypothesis $H_0 : p < p_0$ and alternative hypothesis $H_1 : p \geq p_0$:

$$\psi(\boldsymbol{X}) = \mathbb{I}(L(p = p_0 | \boldsymbol{X}) \leq \alpha) = \mathbb{I}(n > log_{p_0} \alpha) \tag{8}$$

where $\alpha \in [0, 1]$ is the *tolerated error*.

If we consecutively sample n macro-actions, whose subgoals are already known, and $n > log_{p_0} \alpha$, we will assume that a coverage of at least p_0 is achieved. Increasing the desired action coverage p_0 leads to a higher percentage of discovered macro-actions of $\mathcal{M}^*(s_t)$ and to higher quality of each macro-action in $\hat{\mathcal{M}}(s_t)$ at the cost of more trials.

Algorithm 1 Expansion with Macro-Action Generation

```
procedure ExpansionWithMA($\hat{F}, g, t, s_t, H, p_0, \alpha$)
    $n \leftarrow 0$
    repeat                ▷ discover macro actions until confident
        $s_{t+1} \leftarrow s_t$
        $m_t \leftarrow \langle\rangle$
        repeat            ▷ sample until macro state or horizon
            $a_t \sim \mathcal{A}$
            $m_t \leftarrow m_t +\!\!+ \langle a_t \rangle$
            $s_{t+1} \leftarrow \hat{F}(s_{t+1}, a_t)$
        until $(g(s_{t+1}) = 1) \vee (t + |m_t| \geq H)$
        if $\exists m'_t \in \hat{\mathcal{M}}(s_t) : \overline{F}(s_t, m'_t) = s_{t+1}$ then
            $n \leftarrow n + 1$          ▷ macro state rediscovered
            if $\overline{\mathcal{R}}(s_t, m_t) > \overline{\mathcal{R}}(s_t, m'_t)$ then
                $\hat{\mathcal{M}}(s_t) \leftarrow (\hat{\mathcal{M}}(s_t) \setminus \{m'_t\}) \cup \{m_t\}$
        else              ▷ new macro state discovered to expand
            $\hat{\mathcal{M}}(s_t) \leftarrow \hat{\mathcal{M}}(s_t) \cup \{m_t\}$
            return $m_t$
    until $n > log_{p_0} \alpha$
    fullyExpanded($s_t$) $\leftarrow$ true
    return nil
```

4.3 Integration with MCTS

The approach described above can be easily integrated into MCTS, since actions are iteratively explored for each state. Instead of pre-computing the whole set of macro-actions, we only need to find one new macro-action in the expansion step. This saves computation time, since macro-actions are only generated on demand according to the selection policy.

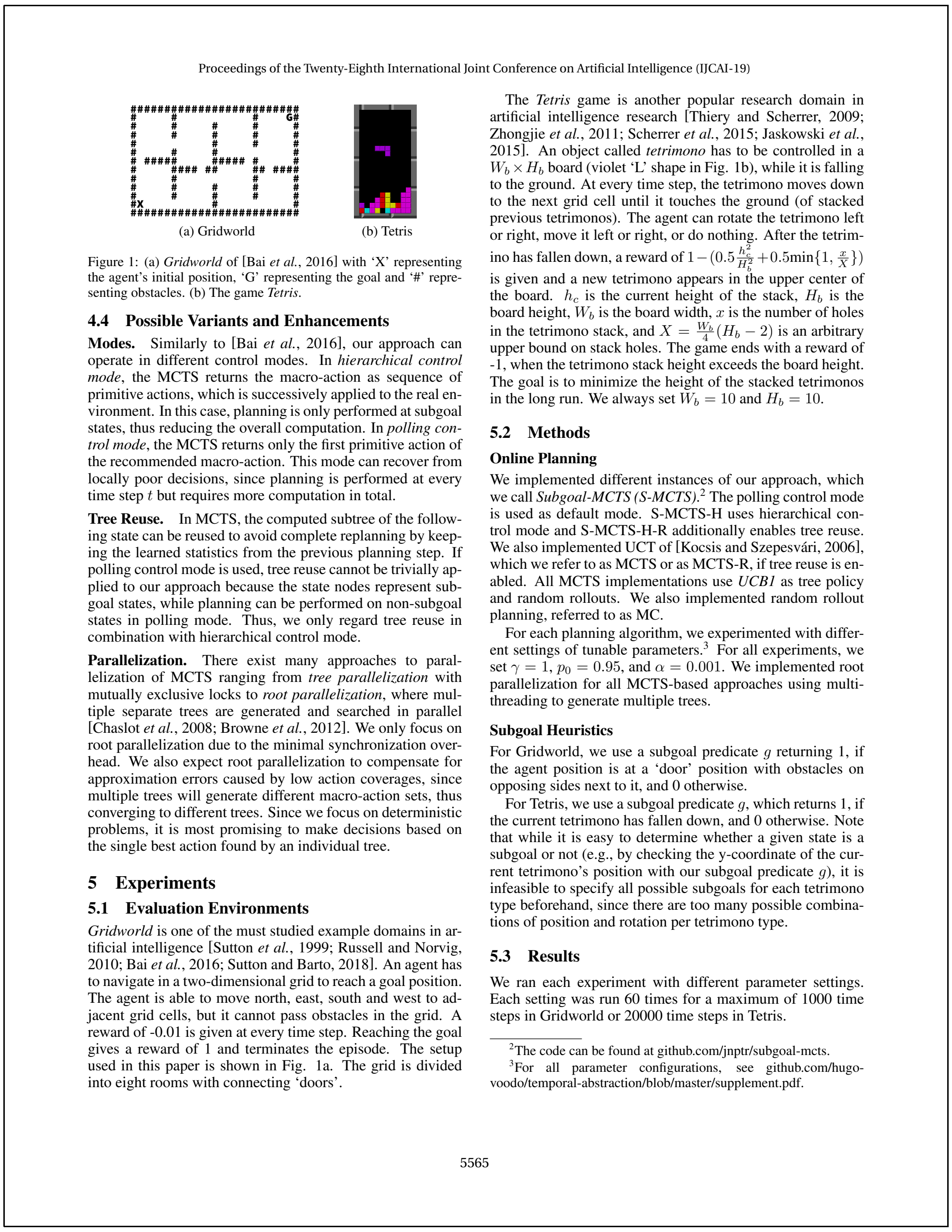
The complete formulation of the modified expansion step is given in Algorithm 1, where $\hat{F}$ is the generative model of the MDP, g is the subgoal predicate, t is the current time step, s_t is the current state, H is the planning horizon, p_0 is the desired action coverage, and α is the tolerated error.

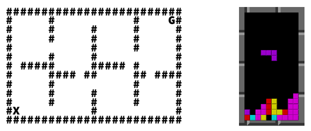
Our approach constructs a search tree for deterministic MDPs, where the root node represents the current state, all other nodes represent subgoal states, and links represent macro-actions. Our expansion step updates the macro-action set $\hat{\mathcal{M}}(s_t)$ for the current leaf node representing s_t. If the desired action coverage p_0 has been achieved, the node of s_t is considered as *fully expanded* and returns *nil*. The expansion step will not be invoked for fully expanded nodes.[1]

The advantage of this approach is that only subgoals *directly reachable* from a given state are regarded. The set of available subgoals per state is only computed *on demand*, when the state is actually visited during tree search, and is not required to be specified beforehand. Our approach enables planning with macro-actions m_t of *arbitrary length* ($1 \leq |m_t| \leq H - t$), thus being more flexible than PUMA [He *et al.*, 2010]. Note that the *Backup* step has to be adjusted to consider the discount of subgoal rewards $\overline{\mathcal{R}}(s_t, m_t)$ according to the length of each macro-action m_t (Eq. 4).

[1]For a complete description of that integration in pseudocode, please refer to github.com/hugo-voodo/temporal-abstraction/blob/master/supplement.pdf.

Taken from original publication: Thomas Gabor, Jan Peter, Thomy Phan, Christian Meyer, and Claudia Linnhoff-Popien. Subgoal-based temporal abstraction in Monte-Carlo tree search. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 5562–5568. AAAI Press, 2019





(a) Gridworld (b) Tetris

Figure 1: (a) *Gridworld* of [Bai *et al.*, 2016] with 'X' representing the agent's initial position, 'G' representing the goal and '#' representing obstacles. (b) The game *Tetris*.

4.4 Possible Variants and Enhancements

Modes. Similarly to [Bai *et al.*, 2016], our approach can operate in different control modes. In *hierarchical control mode*, the MCTS returns the macro-action as sequence of primitive actions, which is successively applied to the real environment. In this case, planning is only performed at subgoal states, thus reducing the overall computation. In *polling control mode*, the MCTS returns only the first primitive action of the recommended macro-action. This mode can recover from locally poor decisions, since planning is performed at every time step t but requires more computation in total.

Tree Reuse. In MCTS, the computed subtree of the following state can be reused to avoid complete replanning by keeping the learned statistics from the previous planning step. If polling control mode is used, tree reuse cannot be trivially applied to our approach because the state nodes represent subgoal states, while planning can be performed on non-subgoal states in polling mode. Thus, we only regard tree reuse in combination with hierarchical control mode.

Parallelization. There exist many approaches to parallelization of MCTS ranging from *tree parallelization* with mutually exclusive locks to *root parallelization*, where multiple separate trees are generated and searched in parallel [Chaslot *et al.*, 2008; Browne *et al.*, 2012]. We only focus on root parallelization due to the minimal synchronization overhead. We also expect root parallelization to compensate for approximation errors caused by low action coverages, since multiple trees will generate different macro-action sets, thus converging to different trees. Since we focus on deterministic problems, it is most promising to make decisions based on the single best action found by an individual tree.

5 Experiments

5.1 Evaluation Environments

Gridworld is one of the must studied example domains in artificial intelligence [Sutton *et al.*, 1999; Russell and Norvig, 2010; Bai *et al.*, 2016; Sutton and Barto, 2018]. An agent has to navigate in a two-dimensional grid to reach a goal position. The agent is able to move north, east, south and west to adjacent grid cells, but it cannot pass obstacles in the grid. A reward of -0.01 is given at every time step. Reaching the goal gives a reward of 1 and terminates the episode. The setup used in this paper is shown in Fig. 1a. The grid is divided into eight rooms with connecting 'doors'.

The *Tetris* game is another popular research domain in artificial intelligence research [Thiery and Scherrer, 2009; Zhongjie *et al.*, 2011; Scherrer *et al.*, 2015; Jaskowski *et al.*, 2015]. An object called *tetrimono* has to be controlled in a $W_b \times H_b$ board (violet 'L' shape in Fig. 1b), while it is falling to the ground. At every time step, the tetrimono moves down to the next grid cell until it touches the ground (of stacked previous tetrimonos). The agent can rotate the tetrimono left or right, move it left or right, or do nothing. After the tetrimino has fallen down, a reward of $1-(0.5\frac{h_c^2}{H_b^2}+0.5\min\{1, \frac{x}{X}\})$ is given and a new tetrimono appears in the upper center of the board. h_c is the current height of the stack, H_b is the board height, W_b is the board width, x is the number of holes in the tetrimono stack, and $X = \frac{W_b}{4}(H_b - 2)$ is an arbitrary upper bound on stack holes. The game ends with a reward of -1, when the tetrimono stack height exceeds the board height. The goal is to minimize the height of the stacked tetrimonos in the long run. We always set $W_b = 10$ and $H_b = 10$.

5.2 Methods

Online Planning

We implemented different instances of our approach, which we call *Subgoal-MCTS (S-MCTS)*.[2] The polling control mode is used as default mode. S-MCTS-H uses hierarchical control mode and S-MCTS-H-R additionally enables tree reuse. We also implemented UCT of [Kocsis and Szepesvári, 2006], which we refer to as MCTS or as MCTS-R, if tree reuse is enabled. All MCTS implementations use *UCB1* as tree policy and random rollouts. We also implemented random rollout planning, referred to as MC.

For each planning algorithm, we experimented with different settings of tunable parameters.[3] For all experiments, we set $\gamma = 1$, $p_0 = 0.95$, and $\alpha = 0.001$. We implemented root parallelization for all MCTS-based approaches using multithreading to generate multiple trees.

Subgoal Heuristics

For Gridworld, we use a subgoal predicate g returning 1, if the agent position is at a 'door' position with obstacles on opposing sides next to it, and 0 otherwise.

For Tetris, we use a subgoal predicate g, which returns 1, if the current tetrimono has fallen down, and 0 otherwise. Note that while it is easy to determine whether a given state is a subgoal or not (e.g., by checking the y-coordinate of the current tetrimono's position with our subgoal predicate g), it is infeasible to specify all possible subgoals for each tetrimono type beforehand, since there are too many possible combinations of position and rotation per tetrimono type.

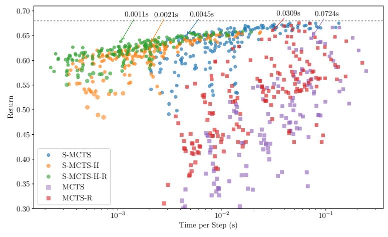
5.3 Results

We ran each experiment with different parameter settings. Each setting was run 60 times for a maximum of 1000 time steps in Gridworld or 20000 time steps in Tetris.

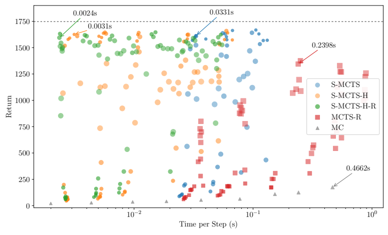
[2]The code can be found at github.com/jnptr/subgoal-mcts.

[3]For all parameter configurations, see github.com/hugo-voodo/temporal-abstraction/blob/master/supplement.pdf.

Taken from original publication: Thomas Gabor, Jan Peter, Thomy Phan, Christian Meyer, and Claudia Linnhoff-Popien. Subgoal-based temporal abstraction in Monte-Carlo tree search. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 5562–5568. AAAI Press, 2019



(a) Gridworld



(b) Tetris

Figure 2: Average return per computation time per step of each planning algorithm using different parameter settings. The x-axis uses a logarithmic scale.

Performance-Computation Tradeoff

First, we evaluated the performance and computation time of each algorithm with different parameter settings (Section 5.2). The results are shown in Fig. 2. Each data point represents the average return for an average amount of computation time per step t given a specific parameter setting. The color indicates the algorithm as described in Section 5.2. The dotted horizontal line indicates the maximum possible return for the respective domain (which overestimates for Tetris).

In Gridworld (Fig. 2a), S-MCTS, S-MCTS-H, and S-MCTS-H-R achieve competitive performance compared to MCTS and slightly lower performance than MCTS-R with much less computation time. In Tetris (Fig. 2b), we omitted MCTS due to intractable run times and replaced it with MC. S-MCTS-H and S-MCTS-H-R quickly achieve near-maximum return with S-MCTS-H-R being less sensitive to the concrete parameter setting. S-MCTS is slower than S-MCTS-H and S-MCTS-H-R but achieves the best overall performance. MCTS-R requires much more computation time than Subgoal-MCTS approaches, while being unable to keep up in performance. MC slightly improves with more computation time but is clearly inferior to all other approaches.

In both domains, S-MCTS-H and S-MCTS-H-R are generally faster than S-MCTS, since computation only takes place at subgoal states. Still, both approaches are able to achieve competitive performance compared to S-MCTS. S-MCTS-H and S-MCTS-H-R also seem to be less sensitive to the parameter settings than the other approaches.

Action Coverage and Parallelization

We also evaluated different combinations of desired action coverages p_0 and tree counts in the Tetris domain. The results are shown in Fig. 3 for S-MCTS, S-MCTS-H, and S-MCTS-H-R. Each plot shows the average return of each algorithm with the best parameter setting (Section 5.2), when using a particular number of trees, which are generated in parallel. The dotted horizontal line indicates the maximum possible return. Increasing p_0 generally leads to increasing return and reduces the variance for all settings. Using a large number of trees can compensate for low desired action coverages p_0, generally leading to higher returns, while requiring less time per step. In case of S-MCTS and S-MCTS-H (Fig. 3a and 3b), all settings achieve similar performance, when the desired action coverage is sufficiently large ($p_0 = 0.95$). S-MCTS-H-R (Fig. 3c) has more variance in its returns, but requires much less time, when using many trees.

Robustness w.r.t. Subgoal Heuristics

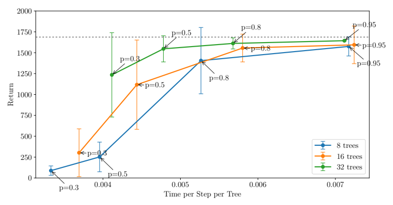
Finally, we evaluated the robustness of our approach w.r.t. different subgoal heuristics for the Gridworld domain with different parameter settings (Section 5.2). The 'exact definition' identifies subgoals by checking, if there are obstacles on opposing sides next to the agent, indicating a 'door' in the grid. A coarser heuristic ($|\mathcal{A}(s_t)| \leq N_a$) checks, if the number of legal actions $|\mathcal{A}(s_t)|$ at the current state is N_a at most. If $N_a = 3$, e.g., then positions next to walls are regarded as subgoals as well. The results are shown in Fig. 4 for S-MCTS, S-MCTS-H, and S-MCTS-H-R. Each data point represents the average return for an average amount of computation time per step t given a specific parameter setting. The color indicates the used subgoal heuristic. The dotted horizontal line indicates the maximum return for the Gridworld domain. When using the heuristic $|\mathcal{A}(s_t)| \leq 2$, then all approaches perform slightly worse than the 'exact definition', while being similarly robust w.r.t. the parameter setting. However, when using the heuristic $|\mathcal{A}(s_t)| \leq 3$, then the performance is generally worse and all approaches are much more sensitive to the parameter setting, while requiring significantly more computation time. Hierarchical control mode and tree reuse seem to slightly improve the robustness of Subgoal-MCTS w.r.t. the parameter setting, while not having a general impact on the overall performance.

6 Discussion

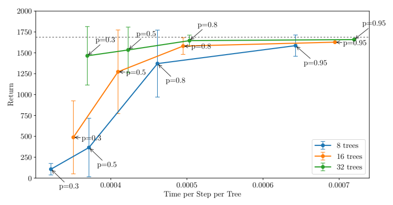
We proposed an approach to general subgoal-based temporal abstraction in MCTS. Our approach approximates a set of macro-actions locally for each state only requiring a generative model and a subgoal predicate.

Our experiments show that S-MCTS and its variants are competitive against standard MCTS in terms of performance-computation tradeoff. While all variants of S-MCTS perform slightly worse in the Gridworld domain, they are able to outperform MCTS-R in Tetris, while generally requiring much less computation time than MCTS in all domains.

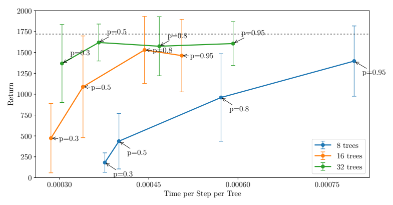
Taken from original publication: Thomas Gabor, Jan Peter, Thomy Phan, Christian Meyer, and Claudia Linnhoff-Popien. Subgoal-based temporal abstraction in Monte-Carlo tree search. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 5562–5568. AAAI Press, 2019



(a) Subgoal-MCTS in polling control mode (S-MCTS)

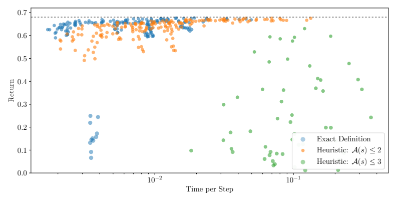


(b) Subgoal-MCTS in hierarchical control mode (S-MCTS-H)

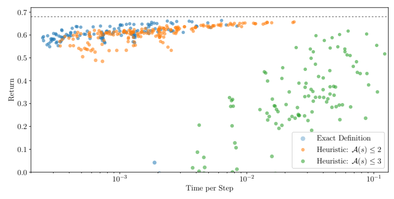


(c) Subgoal-MCTS in hierarchical control mode with tree reuse (S-MCTS-H-R)

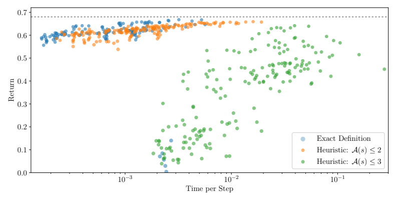
Figure 3: Average return per computation time per step (normalized by the number of trees) of S-MCTS, S-MCTS-H, and S-MCTS-H-R for different desired action coverages p_0 and tree counts in the Tetris domain. The x-axis uses a logarithmic scale.



(a) Subgoal-MCTS in polling control mode (S-MCTS)



(b) Subgoal-MCTS in hierarchical control mode (S-MCTS-H)



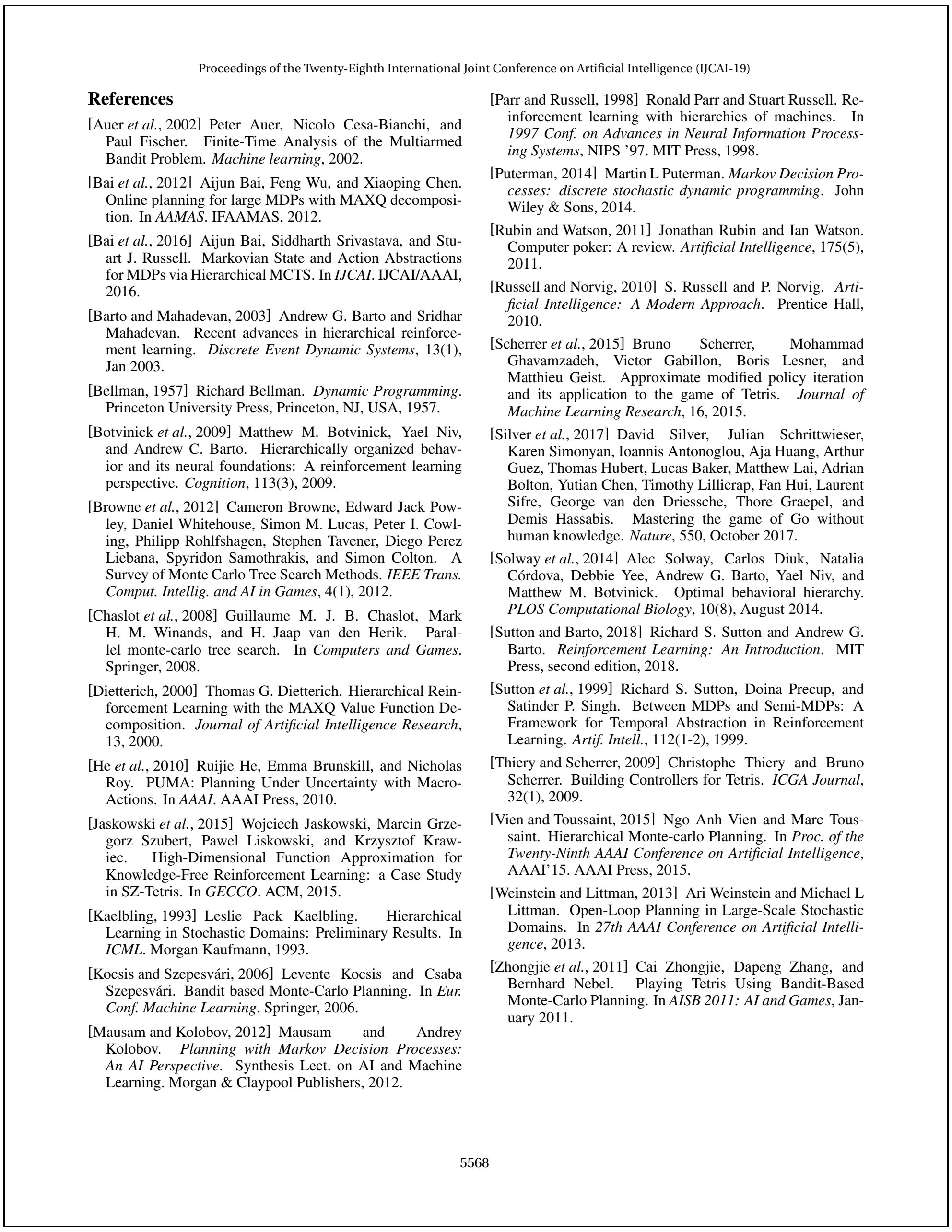
(c) Subgoal-MCTS in hierarchical control mode with tree reuse (S-MCTS-H-R)

Figure 4: Average return per computation time per step of S-MCTS, S-MCTS-H, and S-MCTS-H-R for different subgoal definitions and parameter settings in the Gridworld domain. The x-axis uses a logarithmic scale.

Exploration in S-MCTS depends on the number of subgoals. This was shown in the last experiment, where all variants of S-MCTS displayed worse performance, when using a very coarse subgoal heuristic.When using S-MCTS in hierarchical control mode, computational efficiency can be significantly increased. If a suboptimal choice has been made, S-MCTS-H is unable to recover from the performed actions until the next subgoal is reached, while S-MCTS in polling control mode can locally compensate for suboptimal choices at each time step. Enabling tree reuse slightly improves performance as shown in the first experiment (Fig. 2). Since tree reuse avoids complete replanning, S-MCTS can explore the search space more thoroughly to find better macro-actions. In addition, S-MCTS is shown to benefit from root parallelization. When generating multiple search trees in parallel to search for macro-actions, the performance of all S-MCTS variants can be further improved, while requiring less time. This encourages to exploit multiple cores in real-time applications to make high-quality decisions at certain time frames. When combining tree reuse with a high degree of parallelization, the search time can be drastically reduced by reusing the sets of discovered macro-actions from previous planning steps, while compensating for the approximation errors caused by each individual tree.

Overall, the question of how to adequately define subgoal predicates remains. Future work may further extend flexibility on subgoal predicates, for instance allowing to respect a history of states instead of just a single state.

Taken from original publication: Thomas Gabor, Jan Peter, Thomy Phan, Christian Meyer, and Claudia Linnhoff-Popien. Subgoal-based temporal abstraction in Monte-Carlo tree search. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 5562–5568. AAAI Press, 2019



References

[Auer *et al.*, 2002] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-Time Analysis of the Multiarmed Bandit Problem. *Machine learning*, 2002.

[Bai *et al.*, 2012] Aijun Bai, Feng Wu, and Xiaoping Chen. Online planning for large MDPs with MAXQ decomposition. In *AAMAS*. IFAAMAS, 2012.

[Bai *et al.*, 2016] Aijun Bai, Siddharth Srivastava, and Stuart J. Russell. Markovian State and Action Abstractions for MDPs via Hierarchical MCTS. In *IJCAI*. IJCAI/AAAI, 2016.

[Barto and Mahadevan, 2003] Andrew G. Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(1), Jan 2003.

[Bellman, 1957] Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1957.

[Botvinick *et al.*, 2009] Matthew M. Botvinick, Yael Niv, and Andrew C. Barto. Hierarchically organized behavior and its neural foundations: A reinforcement learning perspective. *Cognition*, 113(3), 2009.

[Browne *et al.*, 2012] Cameron Browne, Edward Jack Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez Liebana, Spyridon Samothrakis, and Simon Colton. A Survey of Monte Carlo Tree Search Methods. *IEEE Trans. Comput. Intellig. and AI in Games*, 4(1), 2012.

[Chaslot *et al.*, 2008] Guillaume M. J. B. Chaslot, Mark H. M. Winands, and H. Jaap van den Herik. Parallel monte-carlo tree search. In *Computers and Games*. Springer, 2008.

[Dietterich, 2000] Thomas G. Dietterich. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Journal of Artificial Intelligence Research*, 13, 2000.

[He *et al.*, 2010] Ruijie He, Emma Brunskill, and Nicholas Roy. PUMA: Planning Under Uncertainty with Macro-Actions. In *AAAI*. AAAI Press, 2010.

[Jaskowski *et al.*, 2015] Wojciech Jaskowski, Marcin Grzegorz Szubert, Pawel Liskowski, and Krzysztof Krawiec. High-Dimensional Function Approximation for Knowledge-Free Reinforcement Learning: a Case Study in SZ-Tetris. In *GECCO*. ACM, 2015.

[Kaelbling, 1993] Leslie Pack Kaelbling. Hierarchical Learning in Stochastic Domains: Preliminary Results. In *ICML*. Morgan Kaufmann, 1993.

[Kocsis and Szepesvári, 2006] Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo Planning. In *Eur. Conf. Machine Learning*. Springer, 2006.

[Mausam and Kolobov, 2012] Mausam and Andrey Kolobov. *Planning with Markov Decision Processes: An AI Perspective*. Synthesis Lect. on AI and Machine Learning. Morgan & Claypool Publishers, 2012.

[Parr and Russell, 1998] Ronald Parr and Stuart Russell. Reinforcement learning with hierarchies of machines. In *1997 Conf. on Advances in Neural Information Processing Systems*, NIPS '97. MIT Press, 1998.

[Puterman, 2014] Martin L Puterman. *Markov Decision Processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[Rubin and Watson, 2011] Jonathan Rubin and Ian Watson. Computer poker: A review. *Artificial Intelligence*, 175(5), 2011.

[Russell and Norvig, 2010] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2010.

[Scherrer *et al.*, 2015] Bruno Scherrer, Mohammad Ghavamzadeh, Victor Gabillon, Boris Lesner, and Matthieu Geist. Approximate modified policy iteration and its application to the game of Tetris. *Journal of Machine Learning Research*, 16, 2015.

[Silver *et al.*, 2017] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550, October 2017.

[Solway *et al.*, 2014] Alec Solway, Carlos Diuk, Natalia Córdova, Debbie Yee, Andrew G. Barto, Yael Niv, and Matthew M. Botvinick. Optimal behavioral hierarchy. *PLOS Computational Biology*, 10(8), August 2014.

[Sutton and Barto, 2018] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, second edition, 2018.

[Sutton *et al.*, 1999] Richard S. Sutton, Doina Precup, and Satinder P. Singh. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artif. Intell.*, 112(1-2), 1999.

[Thiery and Scherrer, 2009] Christophe Thiery and Bruno Scherrer. Building Controllers for Tetris. *ICGA Journal*, 32(1), 2009.

[Vien and Toussaint, 2015] Ngo Anh Vien and Marc Toussaint. Hierarchical Monte-carlo Planning. In *Proc. of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15. AAAI Press, 2015.

[Weinstein and Littman, 2013] Ari Weinstein and Michael L Littman. Open-Loop Planning in Large-Scale Stochastic Domains. In *27th AAAI Conference on Artificial Intelligence*, 2013.

[Zhongjie *et al.*, 2011] Cai Zhongjie, Dapeng Zhang, and Bernhard Nebel. Playing Tetris Using Bandit-Based Monte-Carlo Planning. In *AISB 2011: AI and Games*, January 2011.