Coping with Distance and Location Dependencies in Spatial, Temporal and Uncertain Data

Tobias Emrich

SIGILL·UNIVERSITAT·LUDOVIC·MAXIM

München 2013

Coping with Distance and Location Dependencies in Spatial, Temporal and Uncertain Data

Tobias Emrich

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

vorgelegt von
Tobias Emrich

München, den 08.03.2013

Erstgutachter: Prof. Dr. Hans-Peter Kriegel,
Ludwig-Maximilians-Universität München

Zweitgutachter: Prof. Dr. Thomas Seidl,
Rheinisch-Westfälische Technische Hochschule Aachen

Tag der mündlichen Prüfung: 25.07.2013

Eidesstattliche Versicherung

(Siehe Promotionsordnung vom 12.07.11, § 8, Abs. 2 Pkt. .5.)

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

Name, Vorname

………………………………………… ……………………………………………………

Ort, Datum Unterschrift Doktorand/in

Formular 3.2

Abstract

The amount of collected data nowadays grows in an exponential manner due to a rapid development of capturing (such as photo cameras, environmental sensors or smart phones) and storage devices. Efficiently querying the resulting datasets is a crucial operation in order to retrieve meaningful information and to support time consuming data mining tasks. From a database perspective the efficient processing of queries is further facilitated by two circumstances. First the data becomes more and more complex and is not only given by single values but can amongst others be multidimensional, time-dependent or uncertain. Second the issued queries on these data types themselves need to be more sophisticated since users want to tap the full potential of the available information. This thesis focuses on three complex types of data, namely spatial, uncertain spatial and uncertain spatio-temporal data. For each of these data types certain dependencies are identified which can be utilized for more efficient or more accurate query processing.

The first part of this thesis builds the basis for all following parts. It will give an introduction to the basic concepts of similarity search in databases. Therefore we will review the similarity model based on feature vectors and discuss several similarity query types on multidimensional data. Furthermore we will discuss basic concepts for efficient similarity query evaluation.

During similarity query processing an important step is to filter out true drops as fast as possible. In the second part of this work we introduce the concept of spatial domination which can be utilized for filtering during several similarity queries. State-of-the-art techniques for detecting spatial domination are either not applicable in all scenarios or do not accurately detect it. The reason for the latter problem are so called distance dependencies which are ignored in current methods. Thus this part successively gathers new techniques which overcome the present limitations in different scenarios. At the end of this part a technique is presented which is optimal w.r.t. runtime and accuracy.

In the third part of this thesis the developed techniques from the second part are evaluated in the scope of uncertain spatial data. Again the concept of spatial domination based on distance dependencies can help to filter out objects during query processing. However the nature of uncertain objects requires the techniques for spatial data to be adapted. Thus a method for performing probabilistic domination is introduced. The main issue here is how to set off the single results against each other. For this purpose a technique called “uncertain generating function” is developed and evaluated.

In the last part the focus is changed to uncertain spatio-temporal (UST) data. An

example for UST data are objects which change their positions over time. For the objects the positions are only known for certain points in time (also called observations). Between each two successive observations the position is not exactly known and can only be estimated assuming some kind of model (i.e. the maximum speed of the objects). An analysis of existing works reveals that current state-of-the art models are not able to consider location dependencies of an object between successive time steps. Thus it is also not possible to answer queries according to the possible worlds semantics, which is the prevalent uncertainty model. To overcome this shortcoming a technique based on Markov Chains to model the uncertain movement of objects over time is developed. Utilizing this approach it is possible to efficiently answer queries according to the possible worlds semantics. To support fast query processing even on very large datasets additionally an index structure and an efficient sampling approach are proposed in this part.

Zusammenfassung

Wir beobachten heute, dass die Menge der gesammelten Daten in exponentiellem Maße wächst. Dies ist vor allem auf die schnelle Entwicklung neuer Aufnahmegeräte (wie Fotokameras, Messsensoren oder Smartphones) sowie neuer Speichermedien mit immer größerer Speicherkapazität zurückzuführen. Um nun sinnvolle Informationen und zeitaufwendiges Datamining auf diesen riesigen Datenbeständen zu ermöglichen ist eine effiziente Anfragebearbeitung eine entscheidende Voraussetzung. Aus der Sicht eines Datenbanksystems wird die effiziente Verarbeitung von Anfragen neben der schieren Menge durch zwei weitere Umstände erschwert. Zum einen werden die gesammelten Daten immer komplexer, bestehen also nun nicht nur mehr aus einzelnen Werten, sondern können unter anderem hochdimensional, zeitabhängig oder unsicher sein. Zum anderen werden die Anfragen selbst anspruchsvoller, da Nutzer das volle Potential der vorhandenen Information ausschöpfen wollen. Diese Arbeit konzentriert sich auf drei komplexe Datentypen, nämlich räumliche, unsicher- räumliche und unsichere-raumzeitliche Daten. Für jeden dieser Datentypen werden speziellen Abhängigkeiten identifiziert, welche dann ausgenutzt werden können um Anfragen effizienter oder exakter beantworten zu können.

Der erste Teil der Arbeit bildet die Grundlage für die nachfolgenden Teile und führt die Basiskonzepte von Ähnlichkeitssuche in Datenbanksystemen ein. Dazu wird das Ähnlichkeitsmodel basierend auf Featurevektoren erläutert und unterschiedliche Ähnlichkeitsanfragetypen auf mehrdimensionalen Daten besprochen. Anschließend werden grundlegende Konzepte zur effizienten Bearbeitung von Ähnlichkeitsanfragen eingeführt.

Während der Bearbeitung von Ähnlichkeitsanfragen ist das frühzeitige herausfiltern von Datensätzen, die nicht zum Ergebnis gehören, sehr wichtig um Effizienz zu garantieren. Im zweiten Teil dieser Arbeit wird daher das Konzept „spatial domination“ eingeführt, welches für das Filtern zur Anfragezeit benutzt werden kann. Bisherige Methoden um „spatial domination“ zu erkennen sind entweder nicht allgemein verwendbar oder liefern nicht immer das bestmögliche Ergebnis. Der Grund für das letztgenannte Problem sind sogenannte Distanzabhängigkeiten welche von bisherigen Techniken nicht mit in Betracht gezogen werden. In diesem Teil werden daher nach und nach Techniken entwickelt um die Schwachstellen bisheriger Ansätze zu beseitigen. Zu guter letzt wird ein Verfahren vorgestellt, dass im Hinblick auf Laufzeit und Exaktheit optimal ist.

Im dritten Teil der Arbeit werden die im zweiten Teil entwickelten Techniken auf die Anwendbarkeit im Zusammenhang mit unsicheren-räumlichen Daten hin untersucht. Wie schon zuvor kann das Konzept von „spatial domination“ genutzt werden um möglichst frühzeitig Tupel aus der Ergebnismenge auszuschließen. Jedoch müssen aufgrund der Beschaffenheit von unsicheren Daten die Techniken welche für räumliche Daten entwickelt wurden angepasst werden. Daher wird an dieser Stelle ein Verfahren zur Berechnung von probabilistischer „spatial domination“ entwickelt. Das Hauptproblem an dieser Stelle ist wie einzelne Ergebnisse dieser Berechnungen miteinander verrechnet werden können. Dazu wird die Methode der „uncertain generating functions“ erarbeitet und evaluiert.

Im letzten Teil werden unsichere-raumzeitliche Daten betrachtet. Unsichere-raumzeitliche Daten sind beispielsweise Objekte die Ihre Position über die Zeit hinweg ändern (wie beispielsweise Autos), jedoch nur zu gewissen Zeitpunkten beobachtet werden. Zwischen diesen Beobachtungen ist die Position der Objekte unsicher und kann nur unter Berücksichtigung entsprechender Modelle (wie z.B. der maximalen Geschwindigkeit eines Autos) abgeschätzt werden. Die Analyse von bisherigen Arbeiten auf diesem Gebiet zeigt, dass momentane Modelle keine Ortsabhängigkeiten eines Objektes zwischen aufeinanderfolgenden Zeitpunkten einbeziehen. Dies ist jedoch notwendig um Anfragen unter Berücksichtigung der „possible worlds“-Semantik, welche das vorherrschende Model für Anfragebearbeitung auf unsicheren Daten ist, zu beantworten. Um diese Lücke zu schließen wird ein Verfahren basierend auf Markov Kettenentwickelt um das Verhalten von unsicheren Objekten zwischen Beobachtungen zu modellieren. Mithilfe dieser Technik ist es möglich Anfragen konform zur „possible worlds“-Semantik zu beantworten. Um das Verfahren skalierbar auf große Datenbestände anwendbar zu machen, wird zusätzlich in diesem Teil eine Indexstruktur und eine effizienter Sampling-Ansatz vorgestellt.

Contents

Abstract vii

Zusammenfassung ix

I Preliminaries: Similarity in Spatial Databases 1

1 Introduction 3
1.1 A Data Centric View 3
1.2 Outline 5

2 Similarity Search 7
2.1 Similarity Model 9
2.2 Similarity Query Types 10
2.2.1 Similarity Range Query 10
2.2.2 Nearest Neighbour Query 11
2.2.3 Nearest Neighbour Ranking 12
2.2.4 Reverse Nearest Neighbour Query 13
2.3 Efficient Processing Techniques 14
2.3.1 Index Structures 14
2.3.2 Multi-step Query Processing 16
2.4 Excursion: Reverse Nearest Neighbour Processing 16

II Distance Dependencies in Spatial Data 19

3 Spatial Relations of Approximations 21
3.1 Spatial Domination 22
3.2 Existing Domination Decision Criteria 23
3.2.1 The Hyperplane decision criterion for MBRs 24
3.2.2 The MinMaxDist domination decision criterion for MBRs 24
3.3 Domination Count 26
3.3.1 Partial Domination 27
3.3.2 Boosting Similarity Queries using the Domination Count 28

3.4 Dealing with Distance Dependencies . . . 29

4 The EntryHyperplane Domination Decision Criterion 31

4.1 The Criterion . . . 31
4.2 Domination Count Computation . . . 33
4.3 Application: Incremental Reverse Nearest Neighbour Ranking . . . 36
  4.3.1 Problem Formalization . . . 36
  4.3.2 Related Work . . . 37
4.4 Performing Incremental RNN Ranking . . . 37
  4.4.1 Ranking Count Computation . . . 37
  4.4.2 Best-First Search Based Incremental RNN Ranking Algorithm . . . 39
4.5 Experimental Evaluation . . . 43
  4.5.1 Test Bed . . . 43
  4.5.2 Performance Evaluation . . . 45
  4.5.3 Effect of incorporating Partial Domination . . . 46
  4.5.4 Summary . . . 47
4.6 Conclusions . . . 48

5 The CornerBased Domination Decision Criterion 49

5.1 The Criterion . . . 50
  5.1.1 Geometric Interpretation . . . 50
  5.1.2 Efficient Computation . . . 53
5.2 Application: Constrained Reverse Nearest Neighbor Search on Mobile Objects 54
  5.2.1 Problem Formalization . . . 56
  5.2.2 Related Work . . . 56
5.3 CRkNN Query Processing . . . 57
  5.3.1 General Idea for Answering CRkNN Queries . . . 57
  5.3.2 Pruning Strategies . . . 58
  5.3.3 Refinement Strategies . . . 60
  5.3.4 Algorithm . . . 63
5.4 Experimental Evaluation . . . 63
  5.4.1 Datasets and Parameters . . . 63
  5.4.2 Filter Step . . . 64
  5.4.3 Refinement Strategy . . . 65
  5.4.4 Constraint Parameter m . . . 65
  5.4.5 Bichromatic Case . . . 66
5.5 Conclusions . . . 66

6 The Trigonometric Domination Decision Criterion 69

6.1 Domination Decision Criteria for Spheres . . . 69
  6.1.1 Adopting existing criteria to spherical approximations . . . 70
  6.1.2 The Criterion . . . 71
6.2 Application: All-Nearest-Neighbour Queries . . . 77

6.2.1 Problem Formalization 77
6.2.2 Using Domination Decision Criteria for AkNN 79
6.2.3 Related Work 81
6.3 Performing AkNN Queries 82
6.3.1 Data Structures 83
6.3.2 The AkNN Algorithm 84
6.3.3 Combining Trigonometric Pruning with Other Pruning Criteria . . 85
6.4 Experimental Evaluation 86
6.4.1 Experimental Setup 86
6.4.2 Results 87
6.5 Conclusions 91

7 The Optimal Domination Decision Criterion 93
7.1 The Criterion 94
7.2 Domination Count Computing 98
7.2.1 Partial Domination 99
7.2.2 Domination Count Estimation 101
7.3 Experimental Evaluation 105
7.3.1 Single Object Domination 105
7.3.2 Domination Count Estimation 107
7.3.3 Impact on Standard Spatial Query Processing Methods 107
7.4 Conclusions 109

III Distance Dependencies in Uncertain Spatial Data 111

8 Uncertain Data 113
8.1 Introduction 113
8.2 Modelling Uncertain Data 114
8.2.1 Categorization 114
8.2.2 The Possible Worlds Semantics 115
8.2.3 Uncertain Spatial Data Model 116
8.3 Related Work 118
8.4 Distance Dependencies in Uncertain Spatial Data 119
8.4.1 Probabilistic Domination 119
8.4.2 Probabilistic Domination Count on Uncertain Object Databases . . 122
8.5 Challenges of Distance Dependencies in Uncertain Data 123

9 Probabilistic Domination on Continuous Uncertain Data 125
9.1 Bounding Probabilistic Domination 125
9.2 Bounding the Probabilistic Domination Count 127
9.2.1 Complete Domination 127
9.2.2 Probabilistic Domination 128

9.3 Uncertain Generating Functions (UGFs) 129
9.3.1 Generating Functions 129
9.3.2 Uncertain Generating Functions 131
9.3.3 Efficient Domination Count Approximation using UGFs 132
9.3.4 Generating Functions vs Uncertain Generating Functions 133
9.3.5 Efficient Domination Count Approximation Based on Disjunctive Worlds 136
9.4 Implementation 138
9.5 Applications 139
9.6 Experimental Evaluation 140
9.6.1 Runtime of the Monte-Carlo-based Approach 141
9.6.2 Optimal vs. MinMax Domination Decision Criterion 142
9.6.3 Iterative Domination Count Approximation 143
9.6.4 Queries with a Predicate 143
9.6.5 Number of influenceObjects 144
9.7 Conclusions 144

10 Probabilistic Reverse Nearest Neighbour Queries on Discrete Uncertain Data 147
10.1 Applications of PRkNN Queries 148
10.2 Problem Definition 149
10.2.1 Uncertainty Model 149
10.2.2 PRNN Queries in Uncertain Databases 150
10.3 PRNN Algorithm Sketch 151
10.3.1 Approximation of Objects 152
10.3.2 Spatial Pruning 153
10.3.3 Probabilistic Pruning 153
10.3.4 Verification 153
10.4 Implementation 153
10.4.1 Overview 154
10.4.2 Spatial Pruning 154
10.4.3 Probabilistic Pruning 155
10.4.4 Verification 158
10.5 Probabilistic RkNN Queries 159
10.6 Related Work 160
10.6.1 LC Algorithm 160
10.6.2 CLWZP Algorithm 160
10.6.3 Discussion 161
10.7 Experimental Evaluation 161
10.7.1 Spatial Pruning 163
10.7.2 I/O-Cost 163
10.7.3 CPU-Cost 165
10.8 Conclusions 166

IV Location Dependencies in Uncertain Spatio-Temporal Data 167

11 Uncertain Spatio-Temporal Data 169
- 11.1 Introduction 169
- 11.2 Related Work 170
  - 11.2.1 Interpolation Models 171
  - 11.2.2 Spatio-Temporal Approximation Models 172
  - 11.2.3 Models treating time as a separate dimension 172
  - 11.2.4 Models using stochastic processes 173
- 11.3 Location Dependencies in Uncertain Spatio-Temporal Data 173

12 Querying Uncertain Spatio-Temporal Data 177
- 12.1 Problem Definition 177
- 12.2 Modeling Uncertain Spatio-Temporal Data 179
- 12.3 Probabilistic Spatio-Temporal Query Processing using the Markov-Chain Model 182
  - 12.3.1 Object-Based Query Processing 183
  - 12.3.2 Query-Based Query Processing 185
  - 12.3.3 Discussion 186
- 12.4 Multiple Observations 187
- 12.5 Additional Spatio-Temporal Queries 191
- 12.6 Experimental Evaluation 193
  - 12.6.1 Datasets and Experimental Setting 193
  - 12.6.2 Experiments 194
- 12.7 Conclusion 197

13 Indexing Uncertain Spatio-Temporal Data 199
- 13.1 Problem Definition 200
- 13.2 Related Work 201
- 13.3 Approximating UST-Objects 201
  - 13.3.1 Spatio-Temporal Approximation 201
  - 13.3.2 Spatio-Temporal Filter 204
  - 13.3.3 Probabilistic UST-Object Approximation 205
  - 13.3.4 Finding the optimal Probabilistic Diamond 209
  - 13.3.5 Approximating Probabilistic Diamonds 211
  - 13.3.6 Probabilistic Filter 212
- 13.4 The UST-Tree 214
  - 13.4.1 Architecture 215
  - 13.4.2 Query Evaluation 215
- 13.5 Experimental Evaluation 216
  - 13.5.1 Datasets and Experimental Setting 216
  - 13.5.2 UST-tree Construction 217

13.5.3 Query Performance 218
13.6 Conclusions 222

14 Sampling and Similarity Search on Uncertain Spatio-Temporal Data 223
14.1 Introduction 224
14.2 Problem Definition 226
14.3 Related Work 227
14.4 Uncertain Aligned Longest Common Subsequence 228
14.4.1 Computation of UALCSS 229
14.4.2 Example 232
14.5 Sampling Possible Trajectories 233
14.5.1 Traditional Sampling 234
14.5.2 Adapting the Model to Observations 234
14.5.3 Efficient Model Adaption 235
14.6 Experimental Evaluation 243
14.6.1 Datasets and Experimental Setting 243
14.6.2 Results 246
14.7 Sampling for arbitrary Queries 248
14.8 Conclusion 249

15 Final Remarks 251
15.1 Summary and Contributions 251
15.2 Future Work 253

List of Figures 257

List of Tables 261

References 261

Acknowledgements 281

Curriculum Vitae 283

Part I

Preliminaries: Similarity in Spatial Databases

It is remarkable how similar the pattern of love is to the pattern of insanity.
Andy & Larry Wachowski

Chapter 1

Introduction

1.1 A Data Centric View

We live in a world where the amount of digital data is increasing faster and faster to enormous scales. People are capturing and sharing billions of pictures and videos while companies are recording a plethora of information about their customers, suppliers, and operations. Additionally sensors measuring the position, temperature, humidity, acceleration and much more are included in “intelligent devices”. This phenomenon can be explained by the fact that devices which capture data digitally (and not analogous anymore) are becoming incredibly cheap. Thus almost every private person can afford a digital camera, RFID sensors are attached to throw-away products and all kind of capturing devices are integrated into physical world devices such as mobile phones and automobiles. Furthermore the capacity of storage devices grows in an exponential manner, which allows for keeping most or all of the recorded data. To create order out of this “data monster” and to extract useful information (or knowledge) from the collected data, it is crucial to develop managing and particularly querying techniques which are mainly data driven. This trend has been observed by both industry and research: According to a McKinsey report in May 2011 [117], the analysis of big data has a potential of EUR 180 billion annual value to Europe’s public sector administration. McKinsey projects that 140.000 to 190.000 more positions for deep analytical skills will be available than expected in the U.S. by 2018. In academia this trend towards data-driven research is known as the Fourth Paradigm of scientific research [81]. Although the main scope of this work is similarity search we want to follow this data centric view and structure this thesis according to the considered data types. This proceeding will allow us to give detailed insights on the later defined (distance and location) dependencies which are characteristic for each data type. Particularly we will focus on (multi-dimensional) *spatial data*, which is a very general and important data type as we will discuss in the latter sections. Starting with pure *spatial data* we will include *uncertainty* aspects and the *temporal* dimension afterwards.

Spatial Data

Talking about spatial data one usually thinks about 2 to 3 dimensional position information of geospatial objects for example restaurants, sights, cell phone users or cars. This is an important aspect as services like navigation systems, Google Maps [1] and other location based services enrich our life by valuable information. However in the scope of this work (and many others in this field) spatial data is understood more generally. Through the concept of feature vectors which will be introduced in Section 2.1 it is possible to represent arbitrary objects modelling specific attributes with a high dimensional spatial feature vector. Using these feature vectors as basis it is possible to define distance functions other than the usually used Euclidean distance. Objects which are spatially close according to the used distance function can then be interpreted as similar. This concept is widely implemented in many advanced database systems and allows for a powerful mechanism to express the concept of similarity.

Uncertain Spatial Data

Uncertainty is an inherent attribute of data collected by sensor devices. On the one hand parameters measured by sensors are uncertain due to measurement errors. Tracking a car with the Global Positioning System (GPS) for example only returns a position within several meters of the exact position of the car. On the other hand uncertainty is added due to possibly outdated or non existent information. When the position of a car for example was captured 5 minutes ago and the car then lost the GPS signal due to entering a tunnel or a forest the current position of the car can only be estimated. Simply ignoring this uncertainty, and taking all information as certain facts, does often lead to inconsistent and contradictory data, which can no longer be used for the information gain process. The reason for this degeneration of data is the loss of vital information that is contained in the uncertainty. Proper models treat uncertain values as random variables, and allow us to utilize this information in order to perform meaningful inference.

Uncertain Spatio-Temporal Data

When working with (uncertain) spatial data it is often not only interesting to process the data which is valid at the currrent point of time, but also take a look at historical data or even predicted data. This is obvious since some relations of changing values are only visible considering the temporal context. In such a case the notion of time is crucial. Combining the spatial and the temporal aspect yields interesting and valuable data types such as timeseries (e.g. temperature profile in a city over a year) or trajectories (e.g. planned route of a car). When querying these data types it is particularly important to properly handle the temporal dimension, especially when there are uncertain aspects involved.

---

[1] http://www.maps.google.de

1.2 Outline

This work will investigate several kinds of dependencies which occur during query processing on the three mentioned datatypes. This investigation will lead to better understanding of relations in the considered data type and eventually lead to more efficient or correct query mechanisms. In particular this work is structured as follows:

The rest of Part I of this work (in particular Chapter 2) will give an introduction to the basic concepts of similarity search in databases. Therefore we will review the similarity model based on feature vectors and discuss several similarity query types. Furthermore we will discuss basic concepts for efficient similarity query evaluation.

Part II will focus on spatial data. In Chapter 3 the concept of *spatial domination* is introduced. *Spatial domination* allows us to decide if an object A is definitely closer to object R than another object B. We will show that several similarity queries rely on this relation and thus it is important to efficiently and correctly detect this relation. The main problem here are so called *distance dependencies* which avoid a straightforward computation. The three following chapters (Chapter 4 - 7) each discuss an enhanced method to detect spatial domination (considering the inherent *distance dependencies*). Based on the techniques for *spatial domination* we present methods to compute the *domination count* and show how to integrate this new concept into different similarity query types.

In Part III uncertainty in spatial databases will be considered. Uncertain spatial objects in a database are usually given by a probability density function over the feature space. Since similarity queries on uncertain spatial data also benefit from the concept of spatial domination we investigate the problem of *distance dependencies* under uncertainty in Chapter 8. In particular Chapter 9 shows to compute *probabilistic domination*, which is the equivalent of *spatial domination* incorporating the uncertainty in the data. To compute the *probabilistic domination count* we additionally develop *uncertain generating functions*. The developed techniques are then integrated into the application of reverse-nearest neighbor processing on uncertain data in Chapter 10.

In Part IV we investigate uncertain spatio-temporal (UST) data. Chapter 11 gives an introduction to UST data and reviews related work. The main observation here is that current works do not allow for probabilistic queries (apart from snapshot queries which do not consider the time dimension) since it is not possible to assign confidences to query results. The reason for this shortage is that existing models are not able to handle so called *location dependencies* correctly. In Chapter 12 we propose a model to handle these dependencies correctly and show how the model can be utilized to answer some basic probabilistic spatio-temporal queries. Afterwards an index structure for UST data is developed in Chapter 13 and finally 14 shows how more complicated queries can be answered approximately using an efficient sampling method.

Chapter 2

Similarity Search

Standard Database Management Systems (DBMS) have a long tradition in managing data and retrieving results matching a query which is mostly expressed in SQL (Structured Query Language). Nowadays however exact match and partial-match queries are often not powerful enough for the needs of users in many scenarios. The introduction of similarity opens up various possibilities to query and process the managed data. Similarity search is the task to find objects in a database which are similar to a given query object. Note however that similarity is a concept that is very subjective to each person and always dependent on the context. To illustrate the power of similarity search we will give some sample applications where similarity search is useful and has already been used before.

C ontent based multimedia retrieval: Imagine a user provides a multimedia object and wants to find similar ones in a database. One of the first applications in this direction was the QBIC system [124] by IBM where a user could formulate queries by providing an exemplary image, construct a sketch or select color and texture patterns. The system would then return similar objects from a database filled with images and video content. This area of research has been very flourish in the last two decades not only because the outcome of a research work was mostly fancy. Current systems for image similarity search involve a rather sophisticated pipeline of image pre-processing before the actual search can be performed. This pre-processing is often dependent on the type of images which should be supported. Today's systems are mostly specialized to certain types of content on the images but for those the results of a similarity search are highly reliable. Examples include Google Goggles[1] where a user can take a picture of a building with his cellphone and in return receive background information of this building. Internally the system searches (in a database of images) for the building most probably being identical with the one on the taken picture. Additionally each building in the database is connected to background information of the corresponding building. Another example are face recognition systems as used at airports and for criminal investigations. For audio based similarity search an

---

[1] http://www.google.com/mobile/goggles/

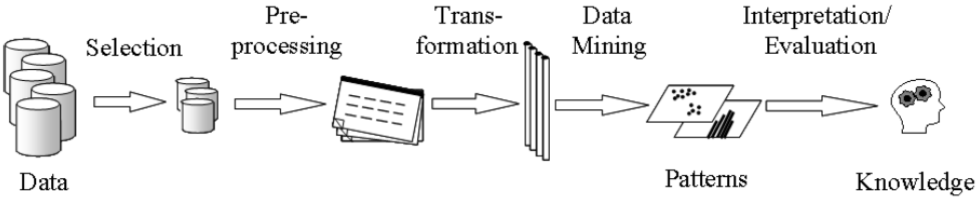


Figure 2.1: The KDD Process

example is Shazam[2], where users can capture several seconds of music with their cell phone and send the recording to the Shazam server. In return they will receive a message containing the interpreter and the title of the song they recorded.

G eo-spatial proximity search: Proximity search in geo-spatial applications is very related to similarity search, since proximity is nothing else than similarity of the positions of two spatial objects. In particular this topic is very interesting for all kinds of location based services. Navigation systems for instance have an integrated function for finding gas stations near the current location of a car. Another example are cell phones which allow for finding restaurants near the current location of the user. Digital Map providers like Microsoft's Bing[3] let the user find businesses in a selected part of the earth.

K nowledge discovery in databases (KDD): KDD is the process of extracting new, valid and potentially useful knowledge from databases. The KDD process, as defined by Fayyad, Piatetsky-Shapiro and [70], has several steps which are depicted in Figure 2.1. The Process always starts with selecting parts of the data base which are relevant. Then preprocessing and possibly transformation is performed. The most important step is the data mining part, which aims at finding patterns which are evaluated by a user to gain knowledge. In this step the automatic detection of information takes place. Data mining includes several subtasks such as clustering, outlier detection and classification. Clustering is the task of grouping objects, where the similarity of objects within a group has to be maximized, while the similarity of objects in different groups has to be minimized. On the contrary outlier detection methods often try to identify objects which are not similar to any object in the database. Consequently clustering and outlier detection methods heavily depend on efficient and effective methods to identify similar or dissimilar objects in the database, or in other words, they depend on similarity search methods. Another data mining task is classification, which aims at assigning to an object the most appropriate class from a set of given classes. An example would be to identify the genre for a given song. Nearest neighbor classifiers are one of several techniques which have shown to be accurate in many scenarios and do rely on the similarity of objects. This shows that similarity search is a technique which can help to boost the performance of several data mining tasks.

---

[2] http://www.shazam.com/

[3] http://www.bing.com/maps

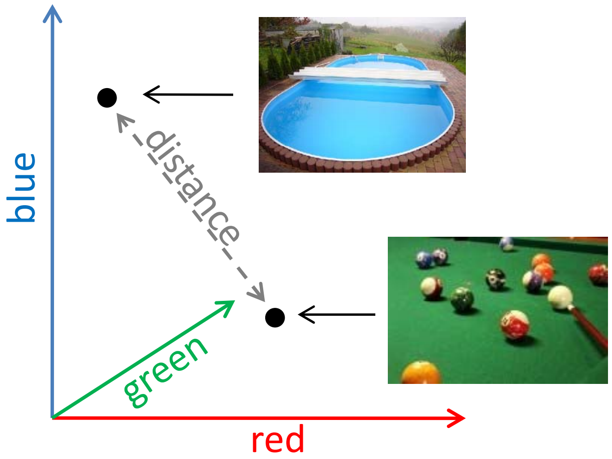


Figure 2.2: (Dis-)similarity between two images in the feature space

Since similarity search is an important aspect in many applications, in the following we will take a detailed look how similarity between arbitrary objects can be modeled, discuss interesting similarity query types and show techniques for efficient retrieval of results.

2.1 Similarity Model

A concept to model similarity is always a tightrope walk between expressiveness and efficiency for query processing. The most prevalent model for similarity search which affords a good trade-off between these two characteristics is the feature-vector based similarity model. Here a domain expert chooses a set of meaningful single-valued numerical features that describe the characteristics of the objects in a database, e.g. size, weight, temperature, etc. This operation is called feature transformation and is of course dependent on the application domain. The selected features span the so-called feature space and after this transformation each object in the database can be represented by one point in this space. This is achieved by interpreting each feature value of an object as a coordinate in the corresponding dimension of the multi-dimensional feature space. Now the distance between two objects can be construed as similarity (small distance) or dissimilarity (large distance) between these objects. The similarity search problem is thus reduced to a proximity search problem which allows us to exploit appropriate spatial access methods and search algorithms. An example for this proceeding is shown in Figure 2.2 where for an image the color components of the RGB color schema (red, green and blue) are selected as features. Since the image showing the pool contains a rather high portion of blue color and the image showing the pool table consists of mostly green pixels, their representations in the feature space reflect this characteristic and consequently the distance between the objects is expected to be relatively large. The remaining problem is to define an appropriate distance measure in the chosen feature space. Formally if O is the universe of all objects that may appear in a database then a similarity distance function is given by:

Definition 2.1 (Similarity distance function)**.** dist : $O \times O \rightarrow \mathbb{R}_0^+$

Several distance measures have been proposed but the most used and prominent are the L_p-norms which are defined as follows:

Definition 2.2 (L_p-norm)**.** Let $x = (x_1, \ldots, x_n), x \in \mathbb{R}^n$ and $y = (y_1, \ldots, y_n), y \in \mathbb{R}^d$ be two vectors in an d-dimensional space. The L_p-norm between x and y is defined as:

$$L_p(x, y) = \sqrt[p]{\sum_{i=1}^{n} |x_i - y_i|^p}$$

For $p = 1$ the Manhattan distance is derived which can be interpreted as the shortest path between two points when moving is only allowed along the edges of an orthogonal grid spanning across the space (like the street network of Manhattan). Setting $p = 2$ corresponds to the well-known Euclidean distance. The Euclidean distance is the most common distance function in many kinds of applications, often because it matches best with the intuition of a user. An important property of an L_p-norm is that it is a metric, which is very desirable for a distance function. Using a metric distance function means that the triangle inequality holds which can often be exploited in order to boost the performance of an application. Besides the basic L_p-norms many other distance measures exist including enhanced versions of the L_p-norms which assign weights to each dimension separately or even incorporate correlation between the feature values of different dimensions through quadratic forms [124]. However in this work we will rely on the basic versions of the L_p-norms.

The feature-vector approach has been successfully used in many application areas ranging from medical imaging [59] over time series [7] and CAD objects [27] to protein databases [93]. Though other possibilities for modeling similarity exist we want to follow this popular approach and assume the feature-based similarity model in the rest of this work.

2.2 Similarity Query Types

The main query predicates from standard DBMS are exact or partial match queries which are not capable to sufficiently express the concept of similarity. For example it is not possible to formulate a query which finds similar objects to a given query object. Thus specialized query types are required for applications involving similarity. In this section, we will present those query types which are most important in similarity search applications. For illustrating these query types we assume that a database $\mathcal{D}$ consisting of N objects $o \in \mathbb{R}^d$ and a distance function *dist* as defined in Definition 2.1 is given.

2.2.1 Similarity Range Query

The most basic similarity query is to find all objects from the database which are closer to a given query object than some threshold according to the similarity distance. This query type is often called range query (RQ) or ϵ-range query with ϵ being the threshold

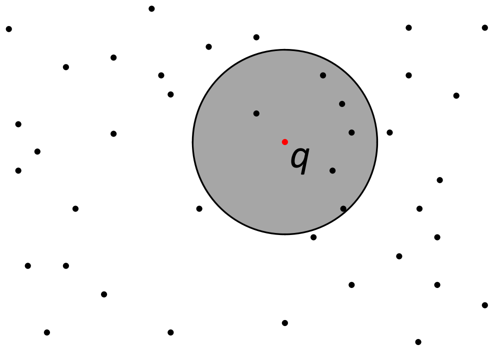


Figure 2.3: Example of a range query with q as query object

parameter. Figure 2.3 illustrates a range query for the query object q in a 2-dimensional feature space assuming Euclidean distance.

Definition 2.3 (ϵ-range query)**.** For a query object $q \in \mathbb{R}^d$ and a query range $\epsilon \in \mathbb{R}_0^+$, the result of a similarity range query is defined as

$$RQ_\epsilon(q) = \{o \in \mathcal{D} | dist(q, o) \leq \epsilon\}$$

Range queries are for example useful for density based clustering algorithms like DBSCAN [68] and OPTICS [9] where an object belongs to a cluster if a certain amount of other objects are located within a predefined range around this object. Thus processing range queries fast also improves the runtime of such KDD algorithms. However the problem with ϵ-range queries is the user-defined threshold value ϵ. Setting ϵ to large yields the whole database as result in the worst case, on the other hand setting ϵ to small may yield no results at all. In many scenarios the parameter ϵ may not be chosen intuitively and in order to get a reasonable result set the user my have to pose several queries with the same query object and different values of ϵ.

2.2.2 Nearest Neighbour Query

Nearest neighbour (NN) queries overcome the problem of an unknown size of the result set. A nearest neighbour query returns the object from the database which has the smallest distance to the query object. A generalisation of nearest neighbor queries are k-nearest neighbor (kNN) queries, which return the k objects from the database with the smallest distance. This means when setting $k = 1$ we end up with an NN query. Figure 2.4 illustrates a kNN query for the query object q and $k = 5$. With this intuitive description we formalize the concept the following way:

Definition 2.4 (k-nearest neighbour query)**.** For a database $\mathcal{D} \subset \mathbb{R}^d$, query object $q \in \mathbb{R}^d$ and a parameter $k \in \mathbb{N}^+$, the result set of a kNN query is defined by

$$kNN(q, \mathcal{D}) = \{o \in \mathcal{D} | \forall o' \in \mathcal{D} \setminus kNN(q, \mathcal{D}) : dist(q, o) <= dist(q, o')\}$$

and $|kNN(q, \mathcal{D})| = k$.

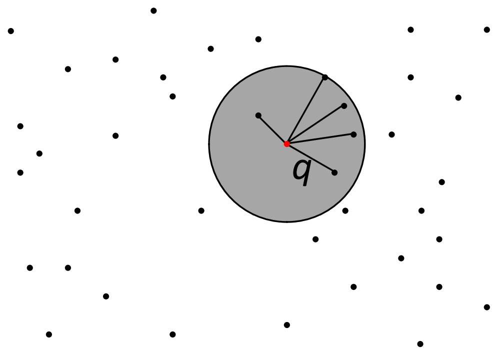
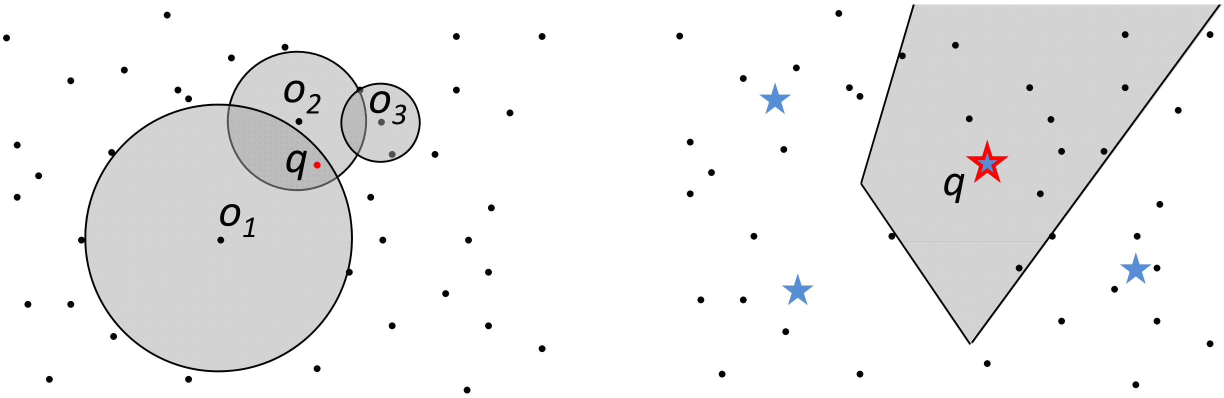


Figure 2.4: Example of a 5NN query with q as query object

Additionally to the 5 nearest neighbours to q Figure 2.4 illustrates the corresponding ϵ-range for a range query returning the same result. In general each kNN query can be transformed into an ϵ-range query returning the same result set. To achieve this the ϵ parameter has to be set to the distance between the query and the furthest object of the result set of the corresponding kNN query. Specifically this is the k^{th}-nearest neighbour and its distance to the query is called the *kNN-distance*. Note that in the deterministic case more than k objects can qualify. However, often the non-deterministic variant is used where exactly k objects are returned and distance ties are broken arbitrarily. Thinking of KDD applications kNN queries are necessary in clustering algorithms like k-means [116] and outlier detection methods like LOF [38]. Again if fast methods for kNN query processing can be developed this will also boost many KDD tasks.

2.2.3 Nearest Neighbour Ranking

One variant of the k-nearest neighbor query is the ranking query that is able to incrementally report the next nearest neighbor of q. Commonly, a *getnext()* function is used to drive the incremental output. The advantage of the ranking query over the ϵ-range query and k-nearest neighbor query is that the user does neither need to choose a similarity threshold ϵ, nor a parameter k specifying the size of the result set. Both attributes can be controlled by performing the ranking and fetching the results iteratively during query time until the user is satisfied. Of course in the most naive implementation this kind of query could be realized by performing a kNN query with increasing k (returning only the new object over the $(k-1)$NN set) each time the *getnext()*-function is called. In this case however the same objects are queried several times which usually results in unnecessary operations. An efficient ranking algorithm should therefore not start over for each *getnext()* call. Nearest neighbor ranking is an important module allowing for more high-level queries like *Top-k* queries [19, 14] or multi-step query processing [141].



(a) Monochromatic R2NN query (b) Bichromatic RNN query

Figure 2.5: The two types of RkNN queries

2.2.4 Reverse Nearest Neighbour Query

In contrast to the three query types above reverse nearest-neighbour queries are a relatively new concept and have first been introduced by Korn and Muthukrishnan [95]. Simply speaking a reverse nearest neighbour (RNN) query returns all objects from a database which have a given query object as nearest neighbour. This definition can straightforwardly be extended to reverse k-nearest neighbour ($RkNN$) queries. Thereby it is distinguished between two types: the monochromatic and the bichromatic case.

In the monochromatic case there is only one set of objects and the query is an object of the same "type" than the other objects (in the database). An example of a monochromatic $R2NN$ query is illustrated in Figure 2.5(a), where the circles corresponding to the 2-nearest neighbour distance of three objects (o_1, o_2, o_3) are drawn. Each object which now has the query object q within this range is in turn R2NN of q (in this example o_1 and o_2). Formally a monochromatic RkNN query is defined by:

Definition 2.5 (monochromatic reverse k-nearest-neighbour query)**.** Given a database $\mathcal{D} \subset \mathbb{R}^d$, a query object $q \in \mathbb{R}^d$ and a parameter $k \in \mathbb{N}^+$, the result set of a monochromatic RkNN query is defined by

$$RkNN^M(q, \mathcal{D}) = \{o \in \mathcal{D} | q \in kNN(o, \mathcal{D})\}$$

For the bichromatic case two sets of objects are required. In the example in Figure 2.5(b) these two sets are given by point objects and the star-shaped objects. The result of a bichromatic RNN query then returns all objects from the point set which have the query object (which is one object of the star set) as nearest neighbour only considering the objects in the star set. In the example all point objects in the grey area are reverse nearest neighbours of the object q. We will show how to efficiently derive these area in the latter Sections. In the general case where $k \geq 1$ a bichromatic reverse k-nearest neighbour query is defined by:

Definition 2.6 (bichromatic reverse k-nearest neighbour query)**.** Given two sets of objects $\mathcal{R}, \mathcal{S} \subset \mathbb{R}^d$, a query object $q \in \mathcal{R}$ and a parameter $k \in \mathbb{N}^+$, the result set of a bichromatic RkNN query is defined by

$$RkNN^B(q, \mathcal{R}, \mathcal{S}) = \{o \in \mathcal{S} | q \in kNN(o, \mathcal{R})\}$$

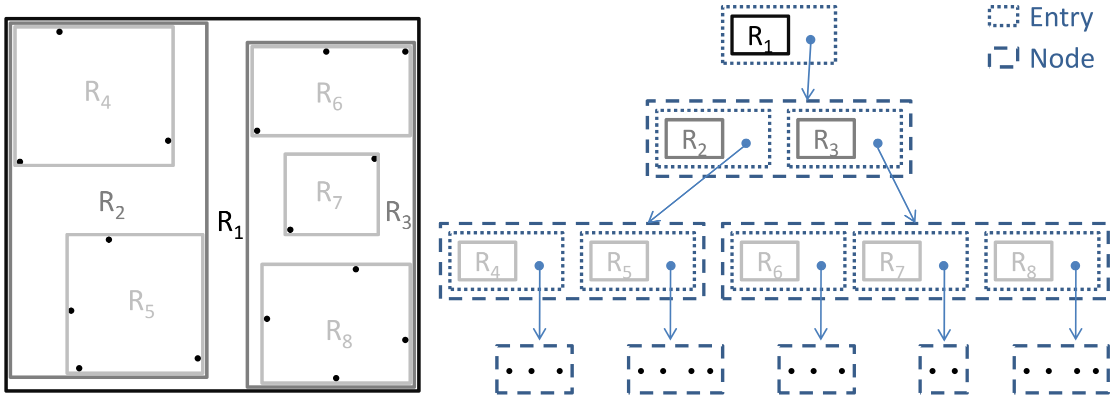
As an example application again consider Figure 2.5(b) and let the objects from the star set be restaurants. Furthermore assume the other objects to be customer locations. Then all customers which are within the gray shaded area have the "query restaurant" as their closest restaurant and are thus likely to be potential guests of this specific restaurant. Appart from that, reverse nearest neighbour queries have a wide range of applications including decision support, continuous referral systems, profile-based marketing, maintaining document repositories and similarity updates in spatial and multimedia databases (cf. [95]). Additionally RkNN queries can be used to speedup several KDD algorithms, for example incremental versions of algorithms relying on the kNN distance. Incremental here means that points are inserted or removed from the database and the data mining algorithm does not have to recompute the result from scratch but use the previous result as input. Examples include incremental versions of the cluster algorithm OPTICS [9] and the outlier detection approach LOF [38]. In this work we concentrate on the monochromatic case if not mentioned otherwise. Note, however that the proposed techniques are usually easily adaptable to the bichromatic case.

2.3 Efficient Processing Techniques

A naive implementation of the ϵ-range and the kNN query with q as query object performs a scan through the whole database computing the distance $dist(q, o)$ to each object $o \in \mathcal{D}$. A ϵ-range query returns all objects having a distance smaller than ϵ, whereas the kNN query returns the k objects with the smallest distance. The runtime complexity (corresponding to the number of distance computations) of these approaches is thus $O(n)$. Things become even worse for the RkNN query where a naive implementation needs to compute a kNN query for each object in the database (which is $O(n)$ in the naive case) and returns the objects having q as one of their k nearest neighbours. This approach yields a complexity of $O(n^2)$. The goal of efficient processing techniques has to be to reduce both CPU cost (mostly consisting of distance computations) and I/O cost (mostly consisting of read operations on the database).

2.3.1 Index Structures

Index structures are a common technique in database systems to speedup queries. They can be divided into two groups: Tree based structures like the B-Tree [20] and Hash based approaches like linear hashing [113]. Though for multi-dimensional vector data there do exist hash based methods [125, 98] the majority of index structures is tree-based. Over the past three decades the variants of the R-Tree family [77] have shown their superiority over



(a) R-Tree visualization (b) R-Tree structure

Figure 2.6: An R-Tree for 2D points

other structures for indexing multidimensional data. The most prominent and prevalent is the R*-Tree [21] but there also exist further extensions like the X-Tree [26] and the IQ-Tree [25] for high dimensional data. The basic idea of all these structures is to group objects in close spatial proximity and bound them spatially using a minimum bounding rectangles (MBR). Repeating this operation recursively on the resulting MBRs yields a hierarchy of MBRs (cf. Figure 2.6(a)). To maintain this hierarchy on the database side (secondary or main memory) a structure using entries and nodes as shown in Figure 2.6(b) is usually used. An entry thereby stores the spatial approximation (in case of an R-Tree a MBR) together with a pointer referring to a node. A node on the other hand contains the entries corresponding to the children of the parent entry according to the hierarchy. The leaf entries (entries at the lowest level of the tree) usually correspond to the multi-dimensional point representation of the objects. A node has a maximum capacity which is chosen such that it utilizes a multiple of the physical block size of the system where the tree is stored. Thus an access to a node corresponds to a random access on the memory which is the main I/O cost factor. For further details about construction and update of R-Trees we refer the interested reader to [21]. Using an R-Tree it is possible to answer many similarity queries very efficiently. The main idea is always to traverse the tree and prune search paths as early as possible only using the MBR approximation of the entries. In this way it is often possible to save distance computations and I/O operations since not all objects of the database have to be considered. A large problem of index structures for multi-dimensional objects however is high dimensionality of feature vectors. This effect is also known as the "curse of dimensionality" [22] and yields a degeneration of the search structure resulting in many unnecessary page accesses and distance computations. Depending on the data distribution it may happen that a naive sequential scan outperforms a multi-dimensional index structure (w.r.t. CPU and I/O) already at around 10 dimensions of the data.

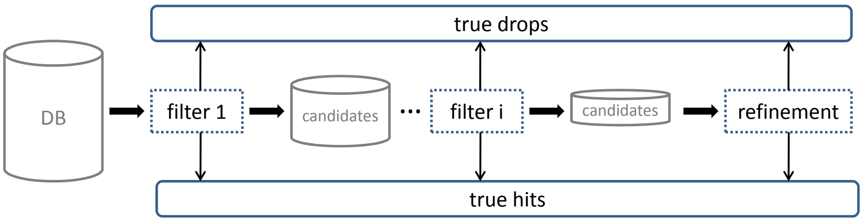


Figure 2.7: Multi-step query processing

2.3.2 Multi-step Query Processing

Particularly when the data types and the defined similarity measure are complex or the data is high dimensional, it is not always possible to index the data for efficient query processing. Under these settings often the sequential scan outperforms index based approaches and another query processing technique may be advantageous: the multi-step approach. The main idea is to replace the costly operations for query processing by much simpler (less costly) ones which indeed are not exact but can be used to remove objects from further consideration. The multi-step approach is depicted in Figure 2.7. It consists of one or more filter steps and one refinement step. Each filter divides the remaining data into one of three categories: Either an object is a true drop[4](can not satisfy the query predicate) , a true hit (satisfies the query predicate) or a candidate (has to be further processed). Each filter in the pipeline should therefore be more selective (usually at higher costs) than the previous one. After filtering the refinement of the remaining candidates has to be performed. This step is usually the most expensive one, but is necessary to yield the final outcome of the query. Examples for the multi-step approach include the VA-File [163] and the optimal index based multi-step kNN algorithm [141] which are both used for similarity search in high dimensional spaces.

Reverse Nearest Neighbour Processing is an excellent example how queries can be answered efficiently using index structures and the multi-step approach. Thus we want to review some existing processing techniques for RNN queries in the following section.

2.4 Excursion: Reverse Nearest Neighbour Processing

The problem of supporting reverse k-nearest neighbor (RkNN) queries efficiently, i.e. computing for a given query q and a number k the RkNNs of q, has been studied extensively in the past years. Existing approaches for RkNN search the Euclidean space can be classified as self-pruning approaches or mutual-pruning approaches.

Self-pruning approaches like the RNN-Tree [95] and the RdNN-Tree [171] are usually designed on top of a hierarchically organized tree-like index structure. They try to estimate the kNN distance of each index entry e which is the maximum kNN distance of one of the objects contained in e. If the kNN distance of e is smaller than the distance

---

[4] The exclusion of objects using a filter is also often called pruning.

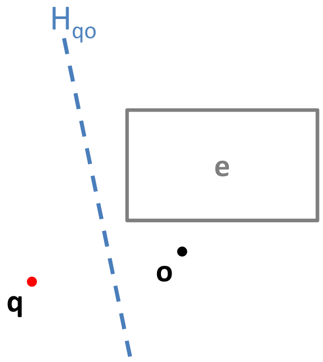


Figure 2.8: TPL pruning ($k = 1$).

of e to the query q, then e can obviousely be filtered as a true drop (pruned). Thereby, self-pruning approaches do not usually consider other entries (database points or index nodes) in order to estimate the kNN distance of an entry e, but simply precompute kNN distances of database points and propagate these distances to higher level index nodes during the construction of the index. Since the kNN distances need to be materialized, these approaches are generally limited to a fixed value of k and cannot be generalized to answer RkNN queries with arbitrary values of k. In addition the performance of approaches based on precomputed distances degenerate when the database is updated frequently.

Mutual-pruning approaches such as [148, 144, 152] use other points to prune a given index entry e. The most general and efficient approach called TPL (by Tao, Papadias and Lian) is presented in [152]. It uses any hierarchical tree-based index structure such as an R-Tree to compute a nearest neighbor ranking (cf Section 2.2.3) of the query point q. The key idea is to iteratively construct Voronoi hyperplanes around q w.r.t. to the points from the ranking. A Voronoi hyperplane is the plane consisting of all points which have equal distance to two points. An example is shown in Figure 2.8 where a Voronoi hyperplane is constructed between the points q and o. Points and index entries that are beyond k Voronoi hyper-planes w.r.t. q can be pruned and need not to be considered for Voronoi construction anymore. Again consider Figure 2.8 for $k = 1$, where the entry e can be pruned, because it is beyond the Voronoi hyper-plane between q and candidate o, denoted by H_{qo}. The reason is that for each object contained in e, the object o is closer than q which in turn means that q cannot be NN of any object in e. For the general case, e can be pruned if e is beyond k hyper-planes w.r.t. all current candidates. If e cannot be pruned, it is refined, or, if e is already a database object, e is a new candidate and the hyperplane H_{qe} will be considered for pruning in the following. If the ranking queue is empty, the remaining candidate points must be refined, i.e. for each of these candidates, a kNN query must be launched. For a more detailed description of the TPL algorithm see [152].

Beside solutions for Euclidean data, solutions for general metric spaces (e.g. [3, 2, 156]) usually implement a self-pruning approach. Typically, metric approaches are less efficient than the approaches tailored for Euclidean data because they cannot make use of the Euclidean geometry.

Part II

Distance Dependencies in Spatial Data

The shortest distance between two points is under construction.
Noelie Altito

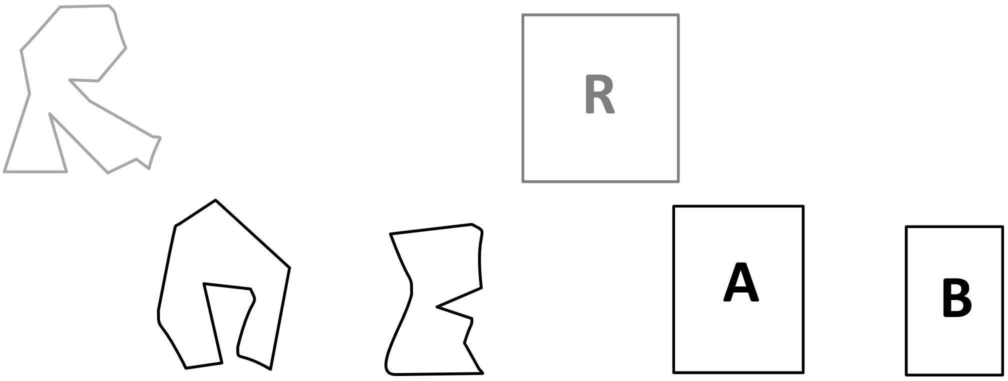
Chapter 3

Spatial Relations of Approximations

During the processing of spatial similarity queries such as distance-range (a.k.a. ε-range) queries, k-nearest neighbor (NN) queries, reverse kNN queries, etc., an important efficiency aspect is early filtering of objects which can not be part of the result set. This filter step is also often called pruning and we will use both terms equivalent in this work if not mentioned otherwise. In this context, spatial pruning techniques are used for numerous application fields including searching in multi-dimensional vector spaces [74, 82, 136], similarity search in time series databases [94], query processing on spatio-temporal data [78, 153] and probabilistic query processing on uncertain data [35, 53, 149]. To perform pruning in the context of spatial applications, the objects are typically approximated by suitable spatial representations like minimum bounding rectangles (MBRs) or bounding spheres (BS) in an early stage of the processing. These approximations are used for different purposes but always aim at saving costly operations. Some exemplary application areas of spatial approximations are the following:

- Approximation of complex spatial objects: Distance computations between complex (high dimensional) polygons (cf. Figure 3.1(a)) are usually very costly. Computing the distance based on the spatial approximations of the polygons (cf. the MBR approximation in Figure 3.1(b)) can yield an upper/lower bound of the exact distance which is efficiently computable. In this case the spatial approximation may safe computational expensive operations.

- Approximation of positions of objects: In a spatio-temporal application, where the position of several cars is captured over time by a central unit. The information of the position of a car at one point of time may be outdated since the car has not sent any new information to the central unit. Instead of polling this information from the car (which usually involves costly usage of a transmission channel) it might suffice to approximate the position by a bounding sphere calculated by taking into account the last known position and the maximum possible speed of the car. In this example the spatial approximation may safe (if the approximation itself yields sufficient information) lots of transmission time during query processing.



(a) Complex 2D Polygons (A', B', R') (b) MBR approximations (A, B, R)

Figure 3.1: Approximations for complex polygons

- Approximation of a set of objects: Tree based spatial index structures like the R-Tree [77] group objects in spatial proximity together in order to enable processing of a whole set of objects during one processing step. In this case approximations allow for saving CPU time (since many objects do not have to be processed) as well as I/O operations (since many objects do not have to be accessed).

These examples show that approximations are ubiquitous in spatial databases. In the following we will introduce two important concepts based on spatial approximations, namely *spatial domination* (cf Section 3.1) and *domination count* (cf Section 3.3) which support efficient query processing. The main problem which prohibits efficient computation of these two concepts are certain *distance dependencies* which we will discuss in Section 3.4. Parts of this chapter have been published in [64] and [60].

3.1 Spatial Domination

Based on this common technique of simple approximation of objects for efficient query processing we introduce the concept of spatial domination. Specifically for three approximations A, B and R (which approximate the objects A', B' and R' respectively) we say A (spatially) dominates B w.r.t. R ($Dom(A, B, R)$) if the following condition holds:

Definition 3.1. $Dom(A, B, R) \Leftrightarrow \forall a \in A, b \in B, r \in R : dist(a, r) < dist(b, r)$

where *dist* is a given distance function defined on multi-dimensional points. In other words $Dom(A, B, R)$ can be used to determine if object A' "is definitely closer to" R' than B' to R'. As we will show in the subsequent sections this relation can be used as a basic operation to support efficient processing of many different similarity queries. For example if $Dom(A, B, R)$ holds then B' cannot be nearest neighbour of R', thus B' can be filtered out (pruned) during NN query processing. However to determine $Dom(A, B, R)$, the right-hand side of the equality from Definition 3.1 is not very helpful because an approximation

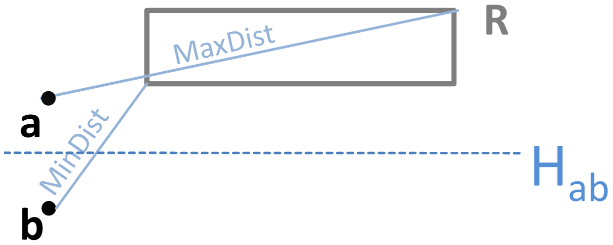


Figure 3.2: MBR pruning example

contains an infinite number of points in $\mathbb{R}^d$ and it is simply not computable to test all triples $a \in A$, $b \in B$ and $r \in R$. Rather, a domination decision criterion $DDC(A, B, R)$ for the domination relation is required, which should fulfill the following properties:

- **Correctness:** if $DDC(A, B, R)$ returns *true* then A dominates B w.r.t. R, i.e.

  $$DDC(A, B, R) \Rightarrow Dom(A, B, R).$$

  This property ensures that pruning based on the domination relation does not generate false drops e.g. objects are excluded from the result set though they belong to the result.

- **Completeness:** if $DDC(A, B, R)$ returns *false* then A does not dominate B w.r.t. R, i.e.

  $$\neg DDC(A, B, R) \Rightarrow \neg Dom(A, B, R).$$

  Though the completeness property is not necessary for most query algorithms to return the correct result set, it is very important in order to exclude as many objects as possible during query processing and thus indispensable for efficient query processing.

- **Efficiency:** $DDC(A, B, R)$ can be evaluated efficiently.

The domination problem is trivial for point objects. However, applied to extended approximations, the domination problem is much more difficult to solve. The problem is that the distance between two objects approximated by rectangles or spheres is no longer a single value but is represented by an interval. If two such distance intervals overlap, we cannot definitely detect whether one distance is smaller than the other.

3.2 Existing Domination Decision Criteria

In this section we will review existing domination decision criteria and analyze them w.r.t. the three properties (completeness, correctness and efficiency).

3.2.1 The Hyperplane decision criterion for MBRs

The TPL-algorithm for RkNN search (cf Section 2.4) provides us with an interesting way for detecting spatial domination. Consider Figure 3.2, where two points a and b and a rectangle R are illustrated. Additionally the dashed Voronoi line H_{ab} between a and b, i.e. the line containing all points that have equal distance to a and b divides the 2D space into two half spaces. The two half spaces are labeled $H_{ab}(a)$ and $H_{ab}(b)$ referring to the half space containing a and b, respectively. It is obvious that all points above that line (located in $H_{ab}(a)$) have an Euclidean distance to a that is smaller than the distance to b. Thus, according to Definition 3.1, a dominates b w.r.t. all objects which lie completely within $H_{ab}(a)$. As a consequence, $Dom(a, b, R)$ holds.

Definition 3.2 (Hyperplane domination decision criterion)**.** Let $a, b \in \mathbb{R}^d$ be points and $R \in \mathbb{R}^d$ be a rectangle. The *Hyperplane* domination decision criterion is defined as

$$\mathrm{DDC}_{HP}(a, b, R) \Leftrightarrow R \subset H_{ab}(a).$$

Lemma 3.1. The *Hyperplane* domination decision criterion is correct, i.e. $\mathrm{DDC}_{HP}(a, b, R) \Rightarrow Dom(a, b, R)$.

Proof. Since for all points $p \in H_{ab}(a)$ it holds that $dist(p, a) < dist(p, b)$ and $R \subset H_{ab}(a)$ it clearly holds: $\forall r \in R : dist(r, a) < dist(r, b) \Rightarrow Dom(a, b, R)$. □

Lemma 3.2. The *Hyperplane* domination decision criterion is complete, i.e. $\mathrm{DDC}_{HP}(a, b, R) \Leftarrow Dom(a, b, R)$.

Proof. If it holds that $\forall r \in R : dist(r, a) < dist(r, b)$ then it is by definition not possible that there exists an $r \in H_{ab}(b)$ thus it has to hold that $\forall r \in R : r \in H_{ab}(a)$. □

The *Hyperplane* domination decision criterion is also efficiently computable when the used distance function is Euclidean distance (L_2-norm). In this case the test if a rectangle is in a half space can be computed in $O(d)$ using simple computational geometry operations [152]. An extension to other L_p-norms is not straightforward.

The most crucial drawback of the Hyperplane domination decision criterion is that a and b are required to be points and not extended approximations such as MBRs. This makes the application of this criterion limited. The next criterion is therefore more general and eliminates this requirement.

3.2.2 The MinMaxDist domination decision criterion for MBRs

Probably the most obvious decision criterion for the domination problem among rectangles is based on two well known metrics defined on rectangles [136]. The minimum distance $MinDist(A, B)$ between two rectangles A and B always underestimates the distance of point pairs $(a, b) \in A \times B$ and is defined as

$$MinDist(A, B) = \sqrt[p]{\sum_{i=1}^{d} \begin{cases} |A_i^{min} - B_i^{max}|^p, & \text{if } A_i^{min} > B_i^{max} \\ |B_i^{min} - A_i^{max}|^p, & \text{if } B_i^{min} > A_i^{max} \\ 0 & \text{, else} \end{cases}} \tag{3.1}$$

where $X_i = [X_i^{min}, X_i^{max}]$ represents the interval of the rectangle X in dimension i and x_i denotes the value of point x in dimension i ($1 \leq i \leq d$).

The maximum distance $\mathit{MaxDist}(A, B)$ between two rectangles A and B always overestimates the distances of all point pairs $(a, b) \in A \times B$ and is defined as:

$$\mathit{MaxDist}(A, B) = \sqrt[p]{\sum_{i=1}^{d} \begin{cases} |A_i^{max} - B_i^{min}|^p & \text{, if } A_i^{mid} \geq B_i^{mid} \\ |B_i^{max} - A_i^{min}|^p & \text{, if } B_i^{mid} > A_i^{mid} \end{cases}} \tag{3.2}$$

where $X_i^{mid} = \frac{1}{2} \cdot (X_i^{min} + X_i^{max})$ is the mean of interval X_i.

Definition 3.3 (MinMaxDist domination decision criterion)**.** Let A, B, $R \in \mathbb{R}^d$ be rectangles. The *MinMaxDist* domination decision criterion is defined as

$$\mathrm{DDC}_{MM}(A, B, R) \Leftrightarrow \mathit{MaxDist}(A, R) < \mathit{MinDist}(B, R).$$

Lemma 3.3. The *MinMaxDist* decision criterion is correct, i.e. $\mathrm{DDC}_{MM}(A, B, R) \Rightarrow \mathit{Dom}(A, B, R)$.

Proof. The following holds due to the conservative properties of *MinDist* and *MaxDist*: $\mathrm{DDC}_{MM}(A, B, R) \Leftrightarrow \mathit{MaxDist}(A, R) < \mathit{MinDist}(B, R) \Rightarrow \forall a \in A, \forall r \in R, \forall b \in B : \mathit{dist}(a, r) \leq \mathit{MaxDist}(A, R) < \mathit{MinDist}(B, R) \leq \mathit{dist}(b, r) \Leftrightarrow \mathit{Dom}(A, B, R)$. □

Lemma 3.4. The *MinMaxDist* decision criterion is not complete, i.e. $\neg\mathrm{DDC}_{MM}(A, B, R) \not\Rightarrow \neg\mathit{Dom}(A, B, R)$.

Proof. Figure 3.2 shows an example for the 2D space where $\mathrm{DDC}_{MM}(A, B, R)$ is false although $\mathit{Dom}(A, B, R)$ holds. In the examples, $A = a$ and $B = b$ are rectangles with zero extension, i.e. points. Clearly, $\mathit{MaxDist}(a, R) < \mathit{MinDist}(b, R)$ is not satisfied, i.e. $\mathrm{DDC}_{MM}(a, b, R)$ is false, although $\mathit{Dom}(a, b, R)$ holds according to the Hyperplane domination decision criterion. □

Here, the problem is that when comparing the maximal distance between A and R with the minimal distance between B and R we take two different positions of the object R into account. For the maximal distance between A and R, we assume that the object R is located at the upper left corner of its rectangle approximation. For the minimum distance between B and R we assume that the object R is located at the lower right corner of its rectangle approximation. However, since an object approximated by a rectangle cannot be located at different positions at the same time, the two distances between A and R and between B and R depend on each other (we call this phenomenon *distance dependencies* and will further discuss this subject in Section 3.4).

Let us note that the *MinMaxDist* domination decision criterion is complete for two arbitrary rectangles A and B if R is a point, i.e. R has no extension in all dimensions. In addition, DDC_{MM} can be computed efficiently in $O(d)$ time since the calculation of *MinDist* and *MaxDist* is linear in d. Additionally the criterion is easily extendable to other approximations than MBRs if *MinDist* and *MaxDist* between each two approximations can be computed (efficiently) [4].

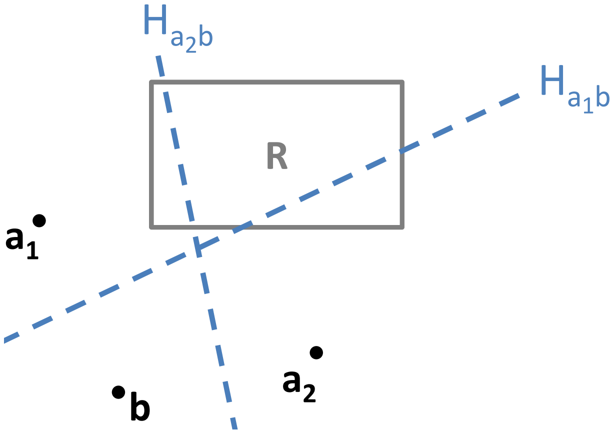


Figure 3.3: a_1 and a_2 are collectively dominating R w.r.t. b

3.3 Domination Count

In most applications, testing the single domination relation of only two approximations (w.r.t. a reference approximation) is too basic. Rather, in the context of a set of approximations $\mathcal{O} \subseteq \mathbb{R}^d$, the number of approximations $A \in \mathcal{O}$ that dominate a given approximation B w.r.t. R (referred to as *domination count*) is required. For example, a kNN query algorithm can use the information that at least k approximations of $\mathcal{O}$ dominate approximation $B \in \mathcal{O}$ w.r.t. a query approximation R to identify B as true drop that can be pruned. The number of approximations that dominate a given approximation can analogously be used e.g. for RkNN queries or inverse ranking queries.

Definition 3.4 (Domination Count)**.** Let $B, R \subseteq \mathbb{R}^d$ be approximations and $\mathcal{O}$ be a set of approximations. The *domination count* of B w.r.t. R is defined by:

$$DC(\mathcal{O}, B, R) = |\{A \in \mathcal{O} | \exists r \in R, \forall a \in A, b \in B : MaxDist(a, r) < MinDist(b, r))\}|$$

Intuitively, if the domination count of B w.r.t. R is k, then for each point $r \in R$ there exist at least k approximations $A \in \mathcal{O}$ which are closer to r than B.

A simple lower bound of the domination count can be achieved by computing the basic domination count. Intuitively, the basic domination count simply counts the number of rectangles that (completely) dominate the rectangle B w.r.t. rectangle R.

Definition 3.5 (Basic Domination Count)**.** Let $\mathcal{O} = \{A_1, ..., A_N\}$ be a set of d-dimensional rectangles and let $B, R \subseteq \mathbb{R}^d$ be two rectangles. The *basic domination count* of B w.r.t. R is the number of objects in $\mathcal{O}$ that dominate B w.r.t. R, formally:

$$DC_{basic}(\mathcal{O}, B, R) = |\{A \in \mathcal{O} \,|\, Dom(A_i, B, R)\}|.$$

This is approach is for example used in [4] and we will refer to it as the naive or basic approach.

3.3.1 Partial Domination

Let us note that the domination count of B w.r.t. R cannot be computed by simply counting the number of approximations that dominate B w.r.t. R by means of Definition 3.5 because this does not involve groups of approximations that dominate R collectively, but not individually. An example of such a group of objects is shown in Figure 3.3. Here a simple example is illustrated consisting of four approximations (note that a point is a special case of a rectangular approximation with 0 extent). Neither point a_1 nor point a_2 dominates the point b w.r.t. R when using the Hyperplane criterion (which is correct and complete). However, B is dominated *partially* by a_1 and partially by a_2, respectively, i.e. it is dominated by a_1 and a_2 w.r.t. specific subregions of R. In other words each point in $r \in R$ is dominated either by a_1 or a_2, which means the domination count of R is 1.

In general we need to find the subregion of R with the minimal domination count, which however is very hard. First, the computation of the intersection of a half-space and a hyper-polyhedron becomes increasingly complex [152] for increasing dimensionality. Secondly, the number of subregions grows very fast. To give a brief intuition of the possible number of subregions generated by a total of n objects, consider the case of axis parallel pruning regions. If $n \leq d$, then each object may split R in a different dimension, resulting in a total of 2^n subregions. For $n > d$, balanced splitting of dimensions results in at least $(1+\lfloor\frac{n}{d}\rfloor^d)$ subregions. If d is assumed to be constant, then $(1+\lfloor\frac{n}{d}\rfloor^d) \in O(n^d)$. Thirdly, the resulting subregions can be complex d-dimensional polygons, particularly the subregions could have not only straight sides but also parabolic sides which makes computations involving these polygons very complex.

Though we are not able to compute the exact domination count of a given rectangle efficiently, we can try to find efficient solutions for approximating the domination count of a rectangle. In principal, in order to determine the domination count of B w.r.t. R we need to take the two constituting types of dominations into account: The first part is to count all objects A for which $Dom(A, B, R)$ holds. This number is called *basic domination count.* The second and more challenging part is to detect all minimal groups $\mathcal{A}$ that dominate B as a group but do not contain an element that already dominates B separately, i.e. each $A_i \in \mathcal{A}$ only partially dominate B. The consideration of this type of domination requires the concept of *partial domination.*

Definition 3.6 (Partial Domination)**.** Let $A, B, R \subseteq \mathbb{R}^d$ be rectangles. *A dominates B partially* w.r.t. R, denoted by $PDom(A, B, R)$ if A dominates B for some, but not all $r \in R$, i.e.

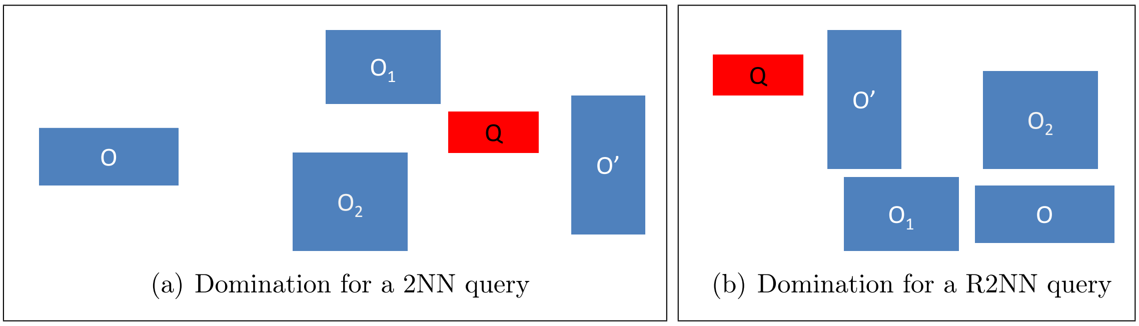
$PDom(A, B, R) \Leftrightarrow$

$$\neg(\forall a \in A, b \in B, r \in R : dist(b, r) > dist(a, r)) \tag{3.3}$$

$$\wedge$$

$$\exists r \in R : \forall a \in A, b \in B : dist(b, r) > dist(a, r) \tag{3.4}$$

Inequality 3.3 holds if A does not dominate B w.r.t. all points $r \in R$. Also note that



(a) Domination for a 2NN query (b) Domination for a R2NN query

Figure 3.4: Domination for pruning

Inequality 3.3 is simply the negation of $Dom(A, B, R)$. In contrast Inequality 3.4 holds if A dominates B w.r.t. at least one point $r \in R$.

3.3.2 Boosting Similarity Queries using the Domination Count

In this section, we will show how the concepts of spatial domination and domination count can be used to boost the pruning power of similarity search algorithms. Following are a few sample query predicates which can be enhanced:

Nearest-Neighbor Search. For a kNN query with query object Q, any object $O \in \mathcal{D}$ can be pruned if $DC(\mathcal{D}, O, Q) \geq k$. Consider Figure 3.4(a) where O is dominated by O_1 and O_2 w.r.t. Q[1]. Thus the *domination count* $DC(\mathcal{D}, O, Q)$ is at least 2. We might as well say:"At least two objects are closer to Q than O". In this case O can safely be pruned from further processing. In contrast O' is not dominated by any other object w.r.t. Q and has to be further processed.

The spatial domination decision criteria can furthermore not only be used for pruning true drops. A problem very similar to pruning is the detection of true hits, i.e. to quickly decide that a potential result candidate must be part of the result set. For example, in the case of kNN queries, an object B is a true hit, if there may be at most k objects that can be closer to R than B. In other words, B is a true hit, if it dominates at least $|\mathcal{D}| - k$ objects. Thus, for a kNN query, an object B is a true hit if $|\{A \in \mathcal{D} | dom(B, A, Q)\}| > |\mathcal{D}| - k$.

Reverse Nearest Neighbor Search. For a general RkNN query with query object Q, any object $O \in DB$ can be certainly pruned if $DC(\mathcal{D}, Q, O) \geq k$. Consider Figure 3.4(b) where Q is dominated by O_1 and O_2 w.r.t. O. Thus the *domination count* $DC(\mathcal{D}, Q, O)$ is at least 2. We might as well say:"At least two objects are closer to O than Q. In this case O can safely be pruned from further processing. In contrast no object dominates Q w.r.t. O' thus it has to be further processed.

As before we are also able to identify true hits. For a RkNN query, an object B is a true hit if $|\{A \in \mathcal{D} | dom(Q, A, B)\}| \geq |\mathcal{D}| - k$.

Inverse Similarity Ranking. The problem of inverse ranking is to determine for a given query object Q the number of objects that are closer to a given reference object R. Such

---

[1] Here we just state that this is the case and not consider this relation in detail.

queries are useful e.g. to determine the financial standing of bank customers in relation to existing customers. In this scenario, the attributes of customers are often uncertain (e.g. income of $40k - 50k$) and thus modeled by uncertain regions, i.e. rectangles. Lower and upper bounds for the rank of Q are $DC(\mathcal{D}, Q, R)+1$ and $|\mathcal{D}|-|\{A \in \mathcal{D} | dom(Q, A, R)\}|+1$, respectively.

3.4 Dealing with Distance Dependencies

The main problem when detecting spatial domination and calculating the domination count is the problem of *distance dependencies*. These *distance dependencies* become obvious when analyzing Definitions 3.1 and 3.4 where we have to verify that $dist(a, r) < dist(b, r)$. Obviously the distance on the left-hand side is dependent on the distance on the right-hand side. This makes the evaluation of these spatial relations problematic. So far we reviewed two domination decision criteria:

- The *MinMaxDist* decision criterion as used in [4]: It is correct and it is efficiently computable as long as the underlying approximations and used distance function allow for efficient computation of MinDist and MaxDist. The main drawback is that it is not complete since it does not consider the aforementioned *distance dependencies* but only uses bounds of the true distances. The domination count can only be naively lower bounded.

- The *Hyperplane* decision criterion as defined in [152]: It is complete, correct and efficiently computable for MBRs of arbitrary dimension if the underlying distance function is the Euclidean distance. The main drawback is that for checking $Dom(A, B, R)$ it the object A and B to be points and not extended approximations. The domination count can be better lower bounded than with the naive implementation (for details we refer to [152]) but this approach scales very bad (regarding the computational cost) for high dimensionality of the data.

In the following chapters of this work we will demonstrate new techniques for efficiently detecting domination, partial domination and calculating the domination count of objects in order to boost different spatial applications. Namely we will introduce

- the *EntryHyperplane* domination decision criterion in Chapter 4: This criterion is an extension of the *Hyperplane* domination decision criterion which is complete and correct for MBRs and the Euclidean Distance. It removes the requirement that for checking $Dom(A, B, R)$, B has to be a point for the cost of a higher computational overhead. Additionally we will show for the case where $d = 2$ how to exactly compute the domination count.

- the *CornerBased* domination decision criterion in Chapter 5: Using this criterion it is possible that all MBR approximations are extended when checking $Dom(A, B, R)$. Still the computational overhead of the *EntryHyperplane* decision criterion remains and the domination count is only naively lower bounded.

- a *Trigonometric* domination decision criterion in Chapter 6: This criterion is designed for spherical approximations when the Euclidean distance is used. Under this setting it is correct, complete and most important efficiently computable. The domination count is again lower bounded naively.
- the *Optimal* domination decision criterion in Chapter 7: Since MBRs are the most prevalent approximation in database applications, this criterion combines the advantages of all before mentioned criteria on MBRs. It is complete, correct, efficiently computable and utilizable under all L_p-norms. The domination count can be iteratively approximated guaranteeing the lower bound property and controlling the computational overhead.

A summary of all domination decision criteria can be found in Table 3.1.

Criterion	Chapter	Complete	Correct	Efficient	Dom. Count	Restrictions
Min-/MaxDist	3	-	✓	O(d)	naive	-
Hyperplane	3	✓	✓	O(d)	costly	L_2, MBR, A, B points
EntryHyperplane	4	✓	✓	$O(2^d)$	optimal (2d)	L_2, MBR, B point
Corner	5	✓	✓	$O(2^d)$	naive	MBR
Spherical	6	✓	✓	O(d)	naive	L_2, Sphere
Optimal	7	✓	✓	O(d)	iteratively	MBR

Table 3.1: Overview over spatial domination decision criteria

Chapter 4

The EntryHyperplane Domination Decision Criterion

In this chapter, we demonstrate how to extend the Hyperplane approach such that for detecting $Dom(A, B, R)$, only B has to be a point (in contrast to the *Hyperplane* domination decision criterion where both A and B have to be points) while still being complete. Additionally we show how based on the concept of *partial domination*, the *domination count* can be efficiently and exactly computed for the case where $d = 2$.

As an application we formalize the novel concept of incremental reverse nearest neighbor ranking. We propose an efficient approach for reporting the results incrementally without the need to restart the search from scratch. We show how the new domination decision criterion and the new domination count computation technique can be incorporated for RNN ranking. Our approach can be applied to a multi-dimensional feature database which is hierarchically organized by any R-tree like index structure. Our solution does not assume any preprocessing steps which makes it applicable for dynamic environments where updates of the database frequently occur. Experiments show that our approach reports the ranking results with much less page accesses than existing approaches designed for traditional reverse nearest neighbor search applied to the ranking problem.

Parts of this chapter have been published in [63]. The reminder of this chapter is organized as follows. In the following Section 4 we introduce the *EntryHyperplane* domination decision criterion. In Section 4.3 we formally define the RNN ranking problem we want to solve here and discuss related work. Section 4.4 explores how the *EntryHyperplane* domination decision criterion can be utilized for RNN ranking. An experimental evaluation is presented in Section 4.5. Last but not least, Section 4.6 concludes the chapter.

4.1 The Criterion

Though complete, correct and efficient DDC$_{HP}$ has the drawback that the objects A and B have to be points and cannot be spatial approximations with extension. In this section we want to show how this constraint can be removed for the object A. Note that to determine if $Dom(A, b, R)$ holds we have to show that each point $r \in R$ is definitely closer to any

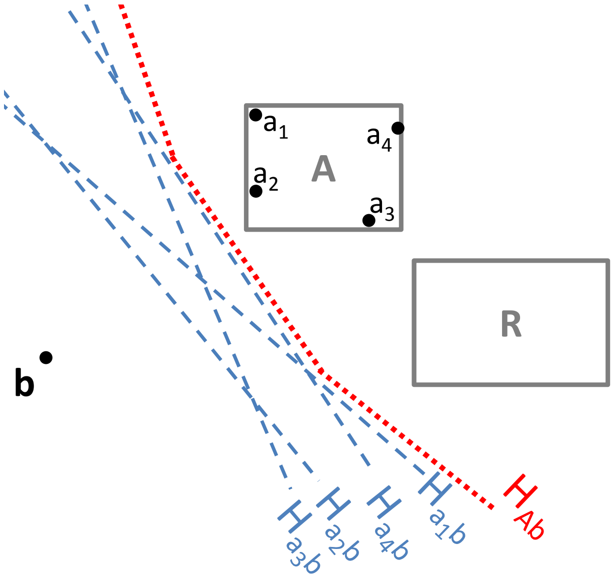


Figure 4.1: The *EntryHyperplane* criterion

point $a \in A$ than to the point b. The basic idea of the new criterion is illustrated in Figure 4.1, where some sample points $a_1, a_2, a_3, a_4 \in A$ are depicted. For each point a_{1-4} we additionally included the corresponding hyperplane $H_{a_{1-4}b}$. Now if we can find a construct H_{Ab} which conservatively (no hyperplane H_{ab_x} intersects with $H_{Ab}(A)$) approximates all hyperplanes build by any point $a \in A$ and additionally use H_{Ab} to check if the object R is on the same side than A ($R \in H_{Ab}(A)$) then we can drop the constraint for A to be a point. Thus for determining $Dom(A, b, R)$ we utilize the Hyperplane domination decision criterion, formally:

$$Dom(A, b, R) \Leftarrow \forall a \in A : \mathrm{DDC}_{HP}(a, b, R)$$

Obviously it is not possible to efficiently use the above formula directly since there exist an infinite number of points $a \in A$. So the task at hand is to find the construct H_{Ab} and use it efficiently.

Figure 4.2 illustrates the computation of such a conservative approximation for a given approximation A in a 2D feature space. First, we have to specify the maximum distance between the mbr-region of the approximation A and any point in the vector space. It suffices to find for each point p in the vector space the point $a \in A$ which is within the mbr-region of A having the maximum distance to p. This can be done by considering partitions of the vector space which are generated as follows: in each dimension the space is split paraxially at the center of the mbr-region of A. As illustrated for the 2D example in Figure 4.2, we obtain partitions denoted by NW, NE, SE and SW. In each of these partitions P, the vertex point of the mbr-region which lies within the diagonal-opposite partition is the mbr-region point which has the maximum distance to all points in P. In our example, for any point p in SW the maximum distance of p to A is the distance between p and point a_{NE} in partition NE. Consequently, the hyperplane $H_{a_{NE}b}$ is a conservative

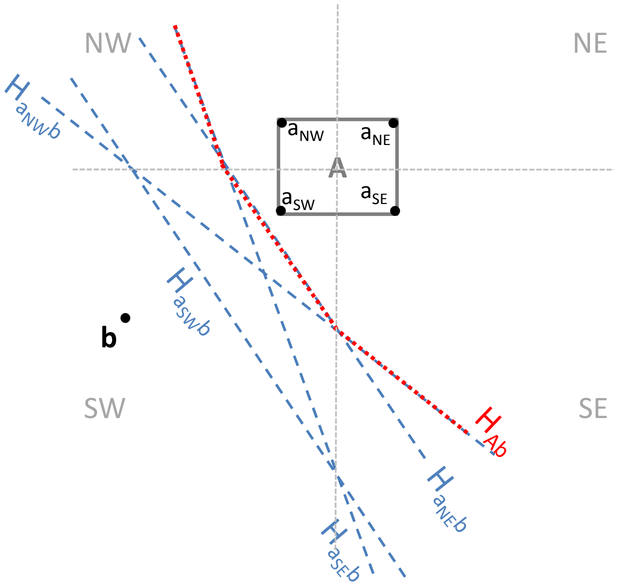


Figure 4.2: Construction of the hyperplane H_{Ab}

approximation of all hyperplanes between points within the mbr-region of A and b within the partition SW. In our example, the complete construct associated with A is composed by the three hyperplanes $H_{a_{NE}b}$, $H_{a_{NW}b}$ and $H_{a_{SE}b}$.

In the general multi-dimensional case H_{Ab} consists of one hyperplane per vertex of the mbr A resulting in 2^d hyperplanes [1]. For checking that an mbr R is on the same side than A of H_{Ab} it is necessary to check if R is on the same side than A for each of these 2^d hyperplanes.

Definition 4.1 (EntryHyperplane Domination Decision Criterion)**.** Let $b \in \mathbb{R}^d$ be a point and $A, R \in \mathbb{R}^d$ be rectangles and let A_v be the set of vertex points of the mbr A. The *EntryHyperplane* domination decision criterion is defined as

$$\mathrm{DDC}_{EH}(A, b, R) \Leftrightarrow \forall a_v \in A_v : R \in H_{a_v b}(A).$$

Lemma 4.1. The *EntryHyperplane* domination decision criterion is complete and correct.

Proof. Follows directly from the construction process of H_{Ab}. Since R has to be located in all $H_{a_v b}(A)$ it is assured that each point $r_i \in R$ is within $H_{a_{v_i} b}(A)$ where a_{v_i} is the vertex point having the largest distance to r_i. □

4.2 Domination Count Computation

Let us note again that the exact domination count $DC(\mathcal{O}, b, R)$ of an object b w.r.t. object R can not be computed by just considering the number of objects $A \in \mathcal{O}$ which fully

---

[1] Let us note that we may ignore the vertex located in the partition containing b

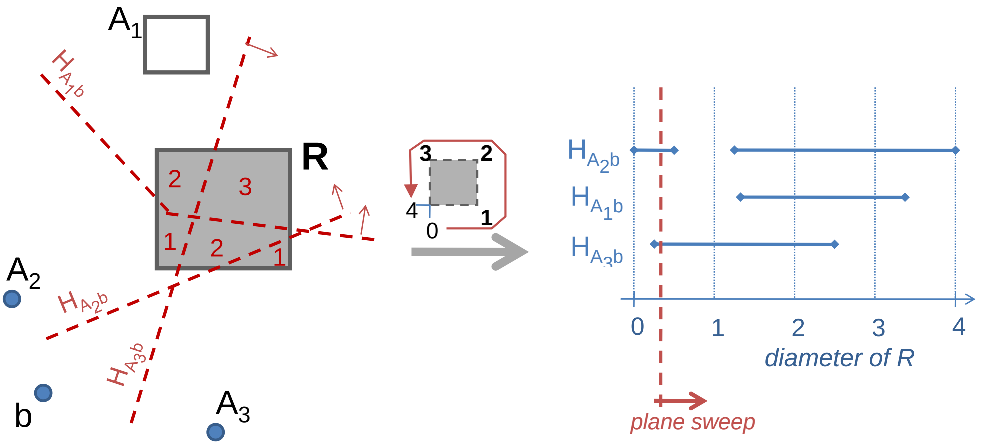


Figure 4.3: Partial pruning based ranking count estimation strategy.

dominate b (cf. Section 3.3). For computing the domination count of b it is necessary to also consider objects which *partially* dominate b. This situation is illustrated in Figure 4.3 where R is intersected by three hyperplanes H_{A_1b}, H_{A_2b} and H_{A_3b} since A_1, A_2 and A_3 partially dominate b w.r.t. R. H_{A_1b}, H_{A_2b} and H_{A_3b} partition R into segments. For any segment s, the domination count of b is equal w.r.t. all points $r \in s$ (illustrated by the red numbers in R). Thus, the minimum number of halfspaces $H_{A_ib}(A_i)$ any point $r \in R$ is covered by, is the minimal domination count of b w.r.t. all segments of R. Therefore, the domination count $DC(\{A_1, A_2, A_3\}, b, R)$ is 1 in our example. However, this computation requires to examine $O(2^m)$ segments, where m is the number of hyperplanes intersecting R. In [152], a similar approach is used, that requires to examine an exponential number of pruning intersection areas. Next, we propose an efficient approach for partial pruning that is based on the following observations:

Lemma 4.2. If the object R contains the point b, then the domination count $DC(\mathcal{O}, b, R)$ is always 0.

The above lemma is obvious, because if R contains b, then R contains one point $r_0 = b$ which has a distance of 0 to b Thus, there can not exist any object $A \in \mathcal{O}$ which has a smaller distance to r_0 than to b.

Lemma 4.3. If an object R does not contain the point b, then the minimal domination count of b w.r.t. all points $r \in R$ is equal to the minimal domination count of b w.r.t. all points on the edges (i.e. the boundary) of R.

Proof. Assume a rectangle R and a point b outside of R (cf. Figure 4.4). Let r be an arbitrary point inside of R and let i denote the intersection of segment $[b, r]$ with the boundary of R. Assume that the domination count of b w.r.t. i is k. Thus, by definition, at least k objects $A_0, \ldots, A_{k-1}$ are closer to i than to b, i.e. $\forall A_j \in \{A_0, \ldots, A_{k-1}\}$: $dist(i, b) > MaxDist(i, A_j)$. For each $A_i \in \{A_0, \ldots, A_{k-1}\}$, the following inequation holds:

$$dist(r, b) = dist(r, i) + dist(i, b) > dist(r, i) + MaxDist(i, A_j)$$

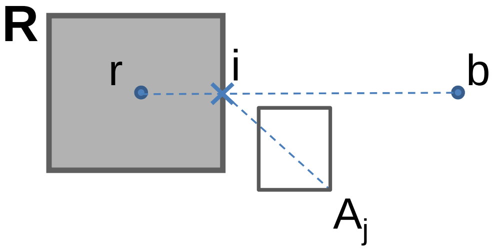


Figure 4.4: Example to Lemma 4.3.

and the triangle inequality yields:

$$dist(r,i) + MaxDist(i, A_j) > dist(r, A_j)$$

Therefore, any object A_j that is closer to i than to b is also closer to r than to b. Thus the domination count of b w.r.t. r is at least the domination count of b w.r.t. i. And since i is located on the boundary of R, $DC(\{A_0, \ldots, A_{k-1}\}, b, r)$ is at least the minimum domination count of b w.r.t. to a point i on the boundary of R. □

For two-dimensional data, we can use the above lemma to efficiently compute the domination count $DC(\mathcal{O}, b, R)$ using the techniques proposed in the previous section for objects which fully dominate b w.r.t. R and the following plane sweep algorithm for the objects which partially dominate b w.r.t. R if b is outside of rectangular region R.

Since only the boundary of an object R is required to determine b's domination count, we linearize the boundary of R to the interval $[0, 4]$. Points on the lower edge of R are represented by their relative position on that edge, points on the right edge of R are presented by the relative position on that edge plus one, and so on (c.f. Figure 4.3). For each hyperplane approximation $H_{A_i b}$ that intersects R, we determine all intersections of $H_{A_i b}$ with R [2] and the respective regions of the boundary of R that is dominated by A_i. In Figure 4.3 b is dominated w.r.t. the boundary of R by A_2 in the interval [2.3,3.4], by A_1 in the interval [1.2,0.5], which is split into two intervals [1.2,4.0] and [0.0,0.5] and by A_3 in the interval [0.2, 2.5]. These intervals can be computed efficiently by simple line-line intersection operations. The task is to find the minimum domination count w.r.t. all points on the boundary of R. The domination count of a point r on the boundary can be obtained by summing up all the domination counts considering all hyperplanes for which the corresponding interval covers r. Now the minimum domination count of of b w.r.t. all points on the boundary can be computed using a plane sweep technique as depicted in Figure 4.3. During the sweep the global minimum of the domination count is computed. This proceeding provides us with an elegant technique for computing $DC(\mathcal{O}, b, r)$. Let us again note that only objects $A_j \in \mathcal{O}$ which do partially dominate b w.r.t. r have to be considered in this plane sweep algorithm. The number objects fulfilling this criteria is usually very small compared to the complete set of objects $|\mathcal{O}|$. Each of these l objects

---

[2] Note that a hyperplane approximation can have at most four intersections with an mbr in $2d$, due to its convex shape.

may generate at most 6 stop points for the plane sweep algorithm, thus the runtime of this proceeding is $O(l)$.

In the following section we will show how to incorporate this technique for computing the domination count can be integrated in order to perform Reverse Nearest Neighbour Ranking.

4.3 Application: Incremental Reverse Nearest Neighbour Ranking

While the reverse nearest neighbor (RNN) search problem, i.e. finding all objects in a database $\mathcal{D}$ that have a given query q among their corresponding k-nearest neighbors, has been studied extensively in the past years, considerably less work has been done so far to support an RNN ranking of objects of a database. An RNN ranking sorts the objects $o \in \mathcal{D}$ of the database according to the number of other objects in the database that are more similar to o than q is. Thus, if an object o has a ranking score of i w.r.t. a query q, object o would also be a reverse k-nearest neighbor of q for all $k \geq i$ but not a reverse k-nearest neighbor of q for all $k < i$.

Initially, the RNN ranking query reports those objects having the smallest ranking scores in a non-deterministic way since several objects may have the same minimal ranking score. Thereby, the results are reported on demand whenever a function called getNext() is invoked. In other words, each consecutive call of getNext() reports one object with minimal ranking score until all objects have been reported.

The major challenge for algorithms that support rankings in general and RNN rankings in particular is that the result of each getNext()-call should be computed incrementally rather than from scratch, i.e. the current state after each getNext()-call needs to be stored and serves as a starting point to compute the results of the next call. The advantage of an incremental ranking method in general is that no parameter k has to be specified for the query in advance and the first (most relevant) results are reported immediately without the overhead of simultaneously computing less relevant results. In addition, if the initial results are not sufficient due to any application specific reasons, further results can be requested on demand by calling the getNext() function.

4.3.1 Problem Formalization

In the following, we assume that $\mathcal{D}$ is a database of n feature vectors and *dist* is the Euclidean distance on the points in $\mathcal{D}$. In addition, we assume that the points are indexed by any traditional aggregate point access method (using rectangles as page approximations) like the aR-Tree family [103, 126].

Here, we will be interested in computing a *ranking* of reverse nearest neighbors (RNNs) w.r.t. a query object q rather than in computing the RkNN of q for a fixed value of k. The concept of the domination count for an object $o \in \mathcal{D}$ from the database helps here. $DC(\mathcal{D} \setminus o, q, o)$ returns the number of objects which are closer to o than the query q. Since the database $\mathcal{D}$ and the query are fixed in this application we will write $DC(o)$ instead of $DC(\mathcal{D} \setminus o, q, o)$. Obviously, it holds that $o \in RkNN(q, \mathcal{D})$ iff $DC(o) \leq k$.

The problem of a *reverse nearest neighbor ranking* is to return incrementally all objects $o \in \mathcal{D}$ in increasing order of the values of $DC(o)$ by calling the method getNext(). In case of ties, getNext() may report any qualifying object, i.e. we will allow for non-determinism. Let us note that the i-th call of getNext() not necessarily return an object that is an RiNN of q, because for a fixed value of k the set $RkNN(q, \mathcal{D}) - R(k-1)NN(q, \mathcal{D})$ generally may contain an (even empty) set of points. In other words, the i-th call of getNext() may report an object o with $DC(o) \neq i$. As a consequence, as additional information, the result of each of the ranking steps should include not only the actual object o but also its *ranking count* (*ranking score*) $DC(o)$.

4.3.2 Related Work

The problem of supporting reverse k-nearest neighbor (RkNN) queries efficiently, i.e. computing for a given query q and a number k the RkNNs of q, has been studied extensively in the past years and has already been discussed in Section 2.4.

Recently, a method for ranked RkNN search has been proposed in [105]. In fact, the authors provide a method for computing the results of an RkNN query with fixed k that are ranked according to k, i.e. the RiNNs are ranked higher than the RjNNs if $i < j \leq k$. This problem is obviously different to the problem of computing an incremental RNN ranking which will be addressed here.

4.4 Performing Incremental RNN Ranking

Our approach is based on an index structure $\mathcal{I}$ for point data which is based on the concept of minimal-bounding-rectangles, e.g. the R-tree family like [77, 21, 26]. In particular, we use multi-resolution aggregate versions of these indexes as described in [103, 126] that e.g. aggregate for each index entry e the number of objects that are stored in the subtree with root e. The set of objects managed in the subtree of an index entry $e \in \mathcal{I}$ is denoted by *subtree*(e). Note that the entry e can be an intermediate node in $\mathcal{I}$ or a point, i.e. an object in $\mathcal{D}$. In the latter case, *subtree*$(e) = \{e\}$.

4.4.1 Ranking Count Computation

The general idea of our solution is based on the TPL algorithm for RkNN search [152], where entries are pruned which are beyond a given number (k) of Voronoi hyperplanes. However, instead of pruning an index entry e, we need the amount of hyperplanes behind which each object $o \in e$ is located w.r.t. q. This corresponds directly to the domination count $DC(\mathcal{D} \setminus o, q, o)$ for all points $o \in$ *subtree*(e). Since computing the exact domination count for an object o requires considering all objects from the database, we will estimate the actual domination count by the function $R(o)$ (denoted as ranking count) during query

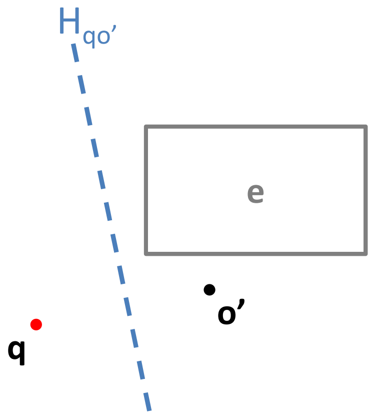


Figure 4.5: Hyperplane decision criterion ($k = 1$).

processing. Specifically $R(o)$ will be set to 0 at the beginning of query processing and will always conservatively lower bound the actual domination count

$$R(o) \leq DC(\mathcal{D} \setminus o, q, o)$$

Additionally during query processing will compute the ranking count of index entries e (denoted as $R(e)$) in the same manner.

Ranking Count Computation based on the Hyperplane criterion

If during query processing we find an object o that dominates q w.r.t. an index entry e, then we know that for all $o' \in \mathit{subtree}(e)$, the value of $R(o')$ can be increased by one. For example, in Figure 4.5, entry e is beyond the Voronoi hyperplane between q and o', denoted by $H_{qo'}$. Thus, o' will have a smaller distance to all objects $o \in \mathit{subtree}(e)$ than q, i.e. all objects $o \in \mathit{subtree}(e)$ will have a ranking count $R(o)$ of at least 1. In this we may also say that the ranking count of e is k (or $R(e) = k$).

Ranking Count Computation based on the EntryHyperplane criterion

Using the Entry Hyperplane criterion we can identify the following relation during query processing: "An index entry e' dominates q w.r.t. an other index entry e" (cf. Figure 4.6). Since we assume that the number of objects stored in the subtree of an index entry e is known, if we exploit the indexing concept as proposed in [103], we also know for the hyperplane associated with that index entry e' ($H_{e'q}$) how many objects this hyperplane relates to. This means that if an entry/object e is behind a hyperplane $H_{e'q}$ associated with an index entry e', the entry e is also behind all hyperplanes H_{oq} associated with the objects $o \in \mathit{subtree}(e')$. We can use this information in order to increase the ranking count of entries according to e' without accessing the child entries of e'. Consequently, the ranking count of an entry/object e which is behind a hyperplane $H_{e'q}$ can be increased by $|\mathit{subtree}(e')|$. In our example $R(e)$ can be increased by 4.

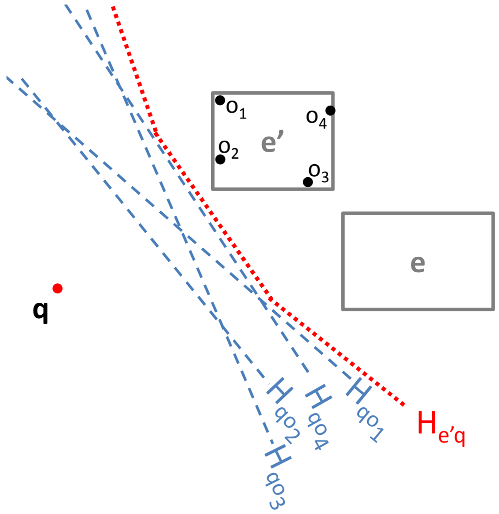


Figure 4.6: Conservative approximation $H_{e'q}$ of the hyperplanes associated with all objects of an index entry e'.

Ranking count computation based on partial domination

Additionally it is possible to further improve the estimation of the exact domination count by using the concept of partial domination (cf. Section 4.2). Since each entry e' in the index has a weight (representing the number of objects which are contained in the entry e') associated with it we have to adapt the proposed technique in the following way (cf Figure 4.7): Since the entry e' is an approximation of $|subtree(e')|$ objects, we may increase the domination count of each segment of e which is fully covered by $H_{e'q}$ by $|subtree(e')|$.

4.4.2 Best-First Search Based Incremental RNN Ranking Algorithm

In this section, we show how we explore the index such that the first results can be reported early without causing unnecessary page accesses. We start with an informal description of our solution before we present implementation details and pseudo code.

Similar to the TPL approach for RkNN queries our approach is based on a best-first search method exploiting a priority queue organizing the index entries to be explored. In contrast to the TPL approach, we propose to give the priorities to the index entries according to the estimated ranking count, i.e. entries with low ranking counts are ranked higher than entries with high ranking count. This means that entries containing objects with a low expected ranking position are explored before entries containing objects with a high expected ranking position. The rational for this strategy is that in this way we try to explore those entries first which contain potential candidates to be reported next from the ranking query.

For the organization of the index entries during the traversal of the index we maintain a priority queue $\mathcal{Q}$ storing entries with the corresponding estimated ranking count which are

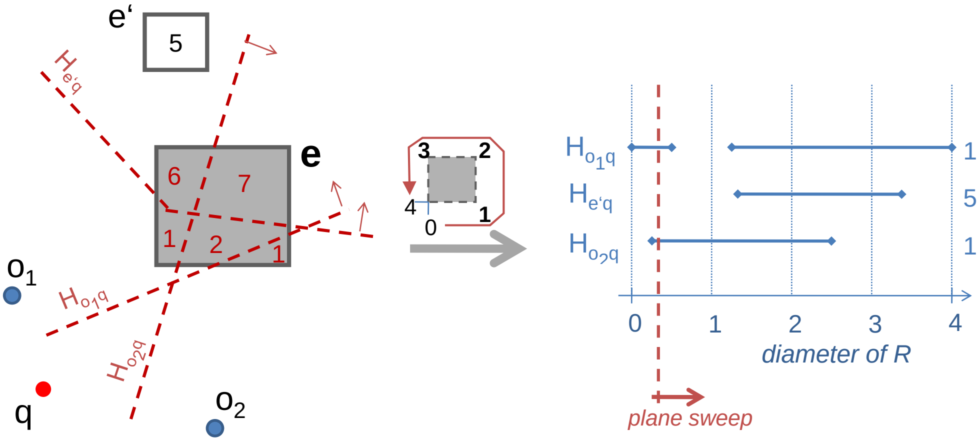


Figure 4.7: Partial pruning for ranking count computation.

sorted in ascending order according to their estimated ranking count. Thereby we assume that the ranking count of each entry in this queue was generated by taking all current entries in the queue into account using the aforementioned strategies for increasing the ranking count.

The top element of the queue is the entry which will be explored next. Whenever an entry e is explored, i.e. e is loaded from disk and is refined, we have to perform the following two steps: first, we update the ranking counts of all elements in the queue according to the children of e and, second, the ranking counts of e's child elements are computed before we insert them into $\mathcal{Q}$.

For the first step, we determine those entries in Q which could be affected by the refinement of an entry e', i.e. for which the ranking count might be increased after refining e'. Obviously, those entries which are completely behind the hyperplane representation of e', $H_{e'q}$, must also be behind the hyperplane representations of each child of e' and, thus, their ranking count is not affected by the refinement of e'. In the example shown in Figure 4.8, entry e_3 is not affected by the refinement of entry e' due to the above considerations. Furthermore, we can ignore those entries e which cannot be behind a hyperplane of any object within $\mathit{subtree}(e')$, e.g. entry e_1 in the example in Figure 4.8, i.e. those entries e for which the following statement holds:

$$\forall p \in e.\mathit{mbr} : \mathit{dist}(p, q) < \mathit{MinDist}(p, e'.\mathit{mbr})$$

which is equivalent to

$$\mathit{Dom}(q, e', e)$$

Since neither the *Hyperplane* nor *EntryHyperplane* criterion are able to detect this relation we rely on the MinMax Decision Criterion (i.e. test if $\mathrm{DDC}_{MM}(q, e', e)$ holds). Let us note that since the DDC_{MM} is not complete this test does not always detect $\mathit{Dom}(q, e', e)$, however since correctness is assured we will not produce any wrong results.

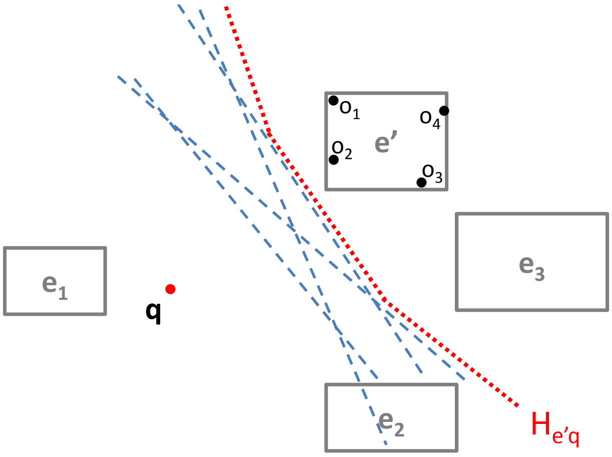


Figure 4.8: Illustration of entries that are/are not affected by the refinement of an entry e.

Corollary 4.1. Entries which may be affected by the refinement of e' are the remaining entries, i.e. those entries e that neither fulfill

$$\mathrm{DDC}_{EH}(e', q, e) \text{ nor } \mathrm{DDC}_{MM}(q, e', e)$$

Each entry e fulfilling none of the above two conditions, e.g. entry e_2 in our example in Figure 4.8, has to be checked against the hyperplane representation of each child of e'. If the entry e is behind the hyperplane representation of a child e'_c of e', its ranking counter will be increased by $|subtree(e'_c)|$.

For the second step, we determine the ranking counts of the children of e'. For that purpose, we simply check all existing entries $e \in \mathcal{Q}$ and all other children of e' whether the current child e'_c of e' is behind the corresponding hyperplanes. If yes, the ranking count of e'_c is increased by $|subtree(e)|$

Finally, if the top entry e in the queue $\mathcal{Q}$ is a point, i.e. $e \in \mathcal{D}$, the point can be output as a result only if neither $\mathrm{DDC}_{EH}(e', q, e)$ nor $\mathrm{DDC}_{MM}(q, e', e)$ for any e' that are currently in the queue. An object e for which we find an entry e' such that the above holds may not have reached its final ranking count (the exact domination count). In this case we need to refine any of those entries $e' \in \mathcal{Q}$ for which this condition holds. As a consequence, e might get a higher ranking count and might be shifted towards the end of $\mathcal{Q}$.

The pseudocode of the algorithm for the incremental RNN ranking is illustrated in Algorithm 1 providing the implementation details of the previously discussed steps. Before the first call of the **getNext()** method, we initialize the priority queue $\mathcal{Q}$ which stores index entries sorted in ascending order of their ranking count. Ties occuring in the priority queue are resolved by, first, prefering leaf index entries to directory index entries and, second, by sorting the entries in increasing distance to q.

The priority queue is initialized with the root of the index. For each call of the **getNext()** method, we dequeue the first entry e of $\mathcal{Q}$. If e is a directory node, then it will be refined calling the **refine()** routine depicted in Algorithm 2. During refinement, we first

Algorithm 1 getNext()

```
while $\mathcal{Q} \neq \emptyset$ do
    $e = \mathcal{Q}$.dequeue()
    if $e$ is a directory entry then
        $refine(e)$
    else
        $e' = refinementRound(e)$
        if $e' = NULL$ then
            return e
        else
            $refine(e')$
        end if
    end if
end while
```

have to find all entries in $\mathcal{Q}$ that are candidates for having their ranking count increased due to the refinement of e. An entry e' is such a candidate, if Corollary 4.1 holds (see line 3 of the refinement procedure in Algorithm 2). These candidates are stored in a list updateI.

Algorithm 2 refine(e)

```
1: updateI= $\{e' \in \mathcal{Q} | \neg\mathrm{DDC}_{EH}(e, q, e') \wedge \neg\mathrm{DDC}_{MM}(q, e, e')\}$
2: updateII= $\{e'' \in (queue \cup result) | \neg\mathrm{DDC}_{EH}(e'', q, e) \wedge \neg\mathrm{DDC}_{MM}(q, e'', e)\}$
3: for all $e_c \in e$ do
4:     for all $e' \in update$I do
5:         if $\mathrm{DDC}_{EH}(e_c, q, e')$ then
6:             increaseRankingCount($e'$, $e_c$.weight)
7:         end if
8:     end for
9:     for all $e'' \in update$II do
10:        if $\mathrm{DDC}_{EH}(e'', q, e_c)$ then
11:            increaseRankingCount($e_c$, $e''$.weight)
12:        end if
13:    end for
14:    for all $e'_c \in e$ do
15:        if $\mathrm{DDC}_{EH}(e'_c, q, e_c)$ then
16:            increaseRankingCount($e_c$, $e_c$'.weight)
17:        end if
18:    end for
19:    $\mathcal{Q}.insert(e_c)$
20: end for
```

Additionally, we need all entries that are candidates for increasing the ranking count of one of the child entries of e. Again we can use Corollary 4.1 to find these candidates (cf line 2 of the **refine()** procedure in Algorithm 2), which are stored in a list updateII. Lines 3-8 check for each child node e_c of e and element $e' \in$ updateI, if e re-ranks e' and increases the ranking count of e' if necessary. Analogously, lines 9-13 increase the ranking count of e_c, if an element $e'' \in$ updateII re-ranks e_c. Then, we increase the ranking count for each child entry e'_c of e that is able to re-rank e_c. Note that e_c and e'_c may be identical, i.e. e_c re-ranks itself. Finally e_c is inserted into the queue $\mathcal{Q}$.

If the entry e is a leaf entry, i.e. e is an object, then e obviously cannot be refined. However, we may not yet return e as a result without further checking, because it may be re-ranked due to an entry that has not yet been refined. In that case, we need to scan the queue $\mathcal{Q}$ for an object that is a candidate for re-ranking e by calling the **refinementRound()** function which is depicted in Algorithm 2 and refining (c.f. Algorithm 2) this object. If no such object exists, e can be returned as the result of the current getNext()-call.

The concept of partial domination for improving the ranking count is not included to keep the pseudo code as simple as possible. It can however be simply integrated at the places in the code where the ranking count of an entry e is updated.

Algorithm 3 refinementRound(e)

```
for all $e' \in \mathcal{Q}$ do
    if $(MinDist(e,e') \leq Dist(e,q) < MaxDist(e,e'))$ then
        return $e'$
    end if
end for
return $NULL$
```

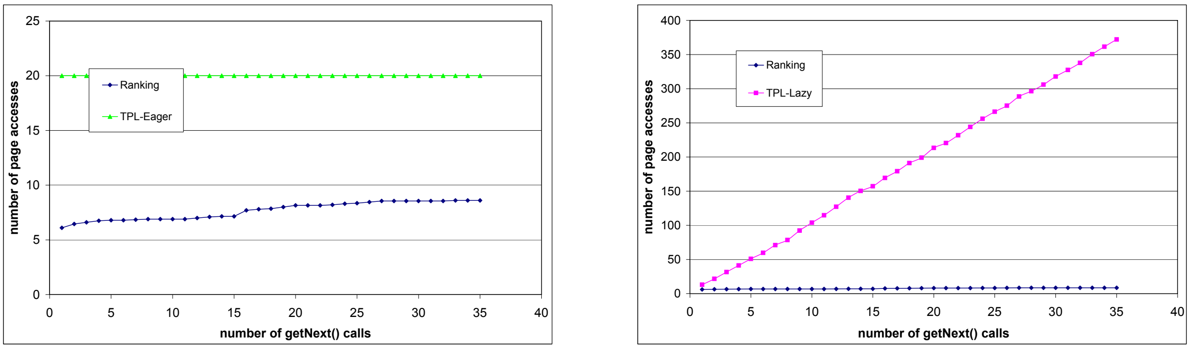
4.5 Experimental Evaluation

In this section, we present the results of our experiments. We start by explaining in detail the settings of our experiments and those of the competitors. Then, we show the results of our performance evaluation on multi-dimensional data. Finally, we evaluate the effect of the partial domination technique on 2D-datasets.

4.5.1 Test Bed

We compared our novel approach for computing an RkNN ranking, with two adaptions of the TPL [152] approach which is the current state-of-the-art algorithm for RkNN query processing. In fact, we applied two versions of the TPL approach for computing a ranking. The problem of the TPL approach is that we cannot predict the number of getNext()-calls beforehand. Thus, we do not know a suitable value of k to answer all getNext()-calls.

The first variant, called TPL-Lazy, implements a lazy strategy assuming that we have a low number of getNext()-calls. It manages a result list which is initially empty and a counter k_c which stores the current value of k and is initialized with $k_c = 1$. The entries in the result list are ordered by increasing ranking scores. For each call of the getNext() method, this variant checks the result list. If the result list is empty, TPL-Lazy computes a RkNN query with $k = k_c$ using the original TPL approach, adds the result of this query to the result list with a ranking score of k_c, and increments k_c. These three steps are processed iteratively until the result list is no longer empty. Last but not least, the TPL-Lazy method returns the next entry in the result list. Obviously, this variant only issues



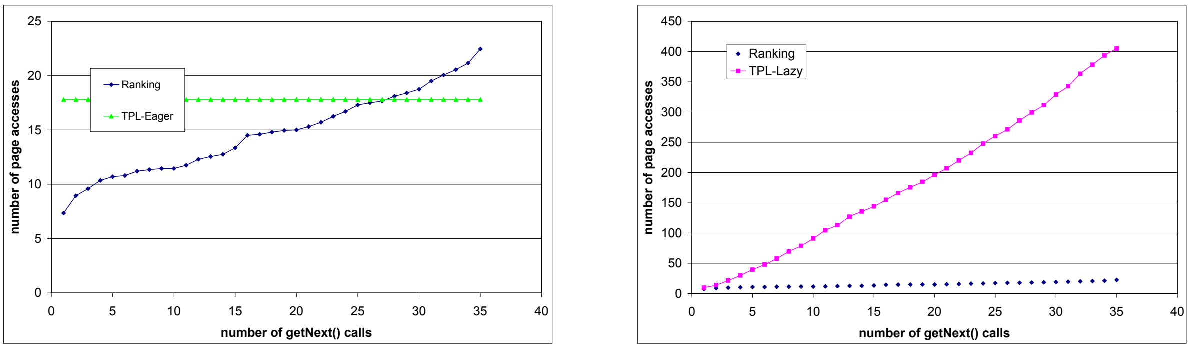
(a) Comparison to TPL-Eager.

(b) Comparison to TPL-Lazy.

Figure 4.9: Comparison of our RkNN ranking with the competitors on uniformly distributed data.

a new RkNN query if necessary beginning with $k = 1$ and successively incrementing the value of k. The costs for answering l getNext()-calls are the sum of the costs of all queries for $k = 1, \ldots$ necessary to answer the l calls.

The second variant, called TPL-Eager, implements an eager policy assuming a higher but possible fixed maximum number of getNext()-calls. It simply assumes that the maximum number of getNext()-calls will be less than the number of result objects of a RkNN query with a special value of k_{max}, e.g. $k_{max} = 100$. Then, we only need to issue one Rk_{max}NN query using the original TPL approach beforehand and sort the results according to their ranking score. Whenever a getNext()-call is issued (and as long as the assumptions stated above regarding the size of the result and the number of getNext()-calls hold), we can simply return the next object from the result list. The costs for answering l getNext()-calls equal to the costs of answering the Rk_{max}NN query (again, as long as the result contains at least l points). Let us note that there is no direct relationship between the number of getNext()-calls l and the value k_{max}. This makes it even harder for the TPL-Eager approach to guess a proper k_{max} value. In fact, to obtain a fair comparison, we computed the most optimistic scenario for the TPL-Eager variant: we first issued l getNext()-calls with our new ranking method and obtained the ranking count of the resulting point of the last call. This count is the optimal k_{max} value for the TPL-Eager approach and we used this value in all our experiments. Thus, in realistic scenarios, the results of a TPL-Eager approach would be worse than presented here. All experiments are based on an aR*-Tree (aggregate version of R*-Tree where each page additionally stores the number of points contained in it [103]) with a page size of 1K. Since all approaches are I/O bound we compared the number of disc pages accessed during the execution of 500 sample RkNN queries and averaged the results.



(a) Comparison to TPL-Eager.

(b) Comparison to TPL-Lazy.

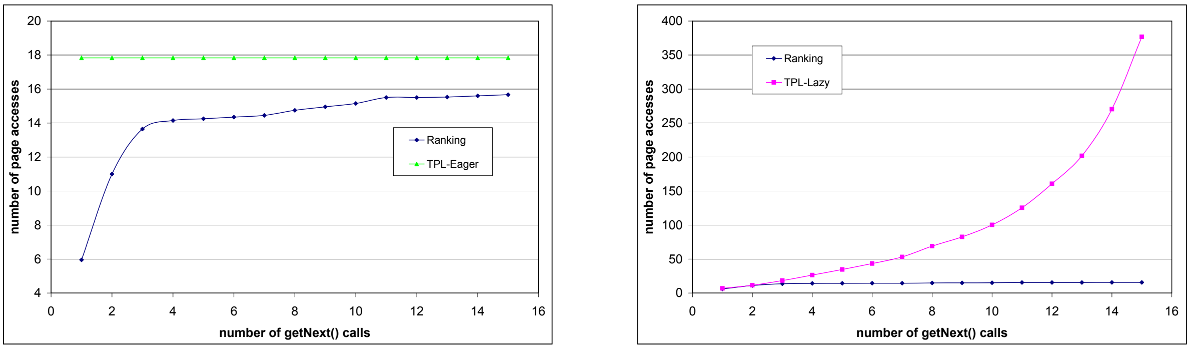
Figure 4.10: Comparison of our RkNN ranking with the competitors on clustered data.

4.5.2 Performance Evaluation

Synthetic Data

We used two synthetic datasets to compare the performance of our ranking algorithm with the two variants of TPL. The first dataset contains 10,000 uniformly distributed 2D points. Figure 4.9 displays the performance of the competitors w.r.t. the number of getNext()-calls. As expected, the performance of the TPL-Eager approach (c.f. Figure 4.9(a))is constant as long as the number of getNext()-calls is smaller than the number of results of the Rk_{max}NN query issued beforehand (which is the case in our scenario – see above). Nevertheless, our ranking algorithm clearly outperforms this TPL variant in terms of query execution times. In fact, the costs of our approach increase only slightly with successive getNext()-calls. In addition, it should be noted that TPL-Eager would need to issue a new RkNN query with a considerably higher value of k if we have more than 35 getNext()-calls because TPL-Eager was optimized for 35 results. Thus, in that case, we would have a jump for the TPL-Eager approach at the 36th getNext()-call while the costs of our ranking algorithm will most likely evolve like in the range of the first 35 getNext()-calls. On the other hand, the costs for the TPL-Lazy variant (cf. Figure 4.9(b)) increase much faster than the costs of our new ranking algorithm. Again our approach clearly outperforms the competitor in terms of query execution times. Note that the performance of our ranking algorithm is of course the same in both Figures 4.9(a) and 4.9(b).

A similar observation can be obtained from Figure 4.10 which displays the performance of the competitors on a 2D synthetic dataset that contains 10,000 points clustered into four different clusters. The only obvious difference is that here, the TPL-Eager approach performs much better than on the uniform dataset. As illustrated in Figure 4.12(a), our ranking algorithm outperforms the TPL-Eager variant only for the first 27 getNext()-calls. In this setting, the TPL-Eager slightly outperforms our ranking algorithm for 30 to 35 getNext()-calls. However, please note that, first, the TPL-Eager approach was implemented with the most optimistic assumptions and can be expected to perform considerably worse in a more realistic scenario where the perfect k_{max} value can usually not be determined



(a) Comparison to TPL-Eager. (b) Comparison to TPL-Lazy.

Figure 4.11: Comparison of our RkNN ranking and the competitors on 5D gene expression data.

beforehand. Second, as explained above, TPL-Eager was optimized for 35 results. If we had more than 35 getNext()-calls, then TPL-Eager would be required again to compute a new RkNN query with a considerably higher value of k which would cause significantly higher costs from the 36th getNext()-call on until the next jump limit is reached.

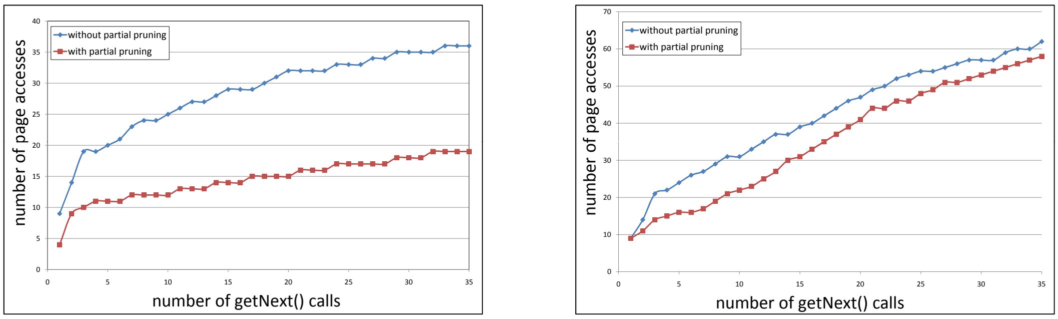
On the other hand, in comparison to the TPL-Lazy variant (cf. Figure 4.12(b)) our ranking algorithm again performs much better and significantly outperforms the competitor in terms of query execution times. Again, the performance of our ranking algorithm is of course the same in both Figures 4.12(a) and 4.12(b).

Real-world Data

We also tested our novel ranking algorithm on real-world data. In Figure 4.11 the performance of our ranking algorithm is compared with the performances of the TPL-Eager variant (cf. 4.11(a)) and the TPL-Lazy variant (cf. 4.11(b)) on a dataset that features the expression level of approx. 6,000 genes under 5 conditions. The result on this 5D dataset is similar to the results on the synthetic datasets reported above. Again, our ranking algorithm clearly outperforms the TPL-Lazy approach. Analogously, the difference to the TPL-Eager approach is less significant but still considerable. It should again be noted that the TPL-Eager variant assumes the most optimistic scenario for its application which is most likely not a realistic setting, and, thus, it can be expected that TPL-Eager performs less accurate in most applications.

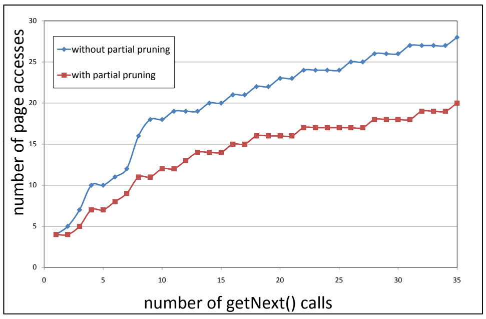
4.5.3 Effect of incorporating Partial Domination

We evaluated our partial pruning technique using a uniformly and a clustered 2D-dataset each containing 10,000 datapoints. The results are depicted in Figure 4.12. Notice that the number of page accesses is reduced by enhancing the domination count estimation by partial domination in both experiments. The effect however, is much more significant on



(a) Uniformly data.

(b) Clustered data.



(c) Real data.

Figure 4.12: Effect of the spatial partial pruning.

the uniformly distributed dataset. Finally, we performed the same experiment on a real world dataset extracted from the Forest Cover Type dataset, retrieved from the UCI KDD repository [13] consisting of 10,000 2D-points [3]. The result in Figure 4.12(c) shows that the spatial partial pruning technique reduces the number of page accesses by about 25%.

4.5.4 Summary

To summarize the results of our experimental evaluation, our novel ranking algorithm outperforms both adaptions of the existing TPL algorithm to the ranking problem, TPL-Eager and TPL-Lazy, significantly in terms of query execution times. While TPL-Eager seems to be competitive (if at all) only for a higher number of getNext()-calls, TPL-Lazy seems to be competitive (if at all) only for a very low number of getNext()-calls. This result is quite intuitive because TPL-Eager tries to estimate the worst-case by precomputing the maxi-

---

[3] Here we only used the first two dimensions corresponding to spatial locations and the first 10,000 feature vectors

mum number of required results for a maximum number of getNext()-calls by computing one Rk_{max}NN query. Thus, the more the number of getNext()-calls reaches the number of resulting objects of the Rk_{max}NN query, the more the costs for the Rk_{max}NN query pay off. Otherwise, TPL-Eager caused a large portion of unnecessary costs to compute a large number of results that are not needed. On the other hand, TPL-Lazy assumes the best case of very few getNext()-calls and, thus, computes results only if necessary by consecutively issuing a RkNN query with increasing k. Obviously, as long as the number of consecutive RkNN queries, with increasing k necessary to report results, is small, i.e. the number of getNext()-calls is low, this strategy pays off. Otherwise, TPL-Lazy constantly recomputes RkNN queries with the next higher value for k which produces a lot of redundant results w.r.t. the previously computed queries.

Our ranking algorithm obviously performs best because it does not assume worst- or best-cases but focuses on computing the ranking incrementally. Since in a ranking query scenario, it is not known beforehand, how often the method getNext() is called, this is the most efficient solution in the general case but also – as our experiments illustrate – in the borderline cases where either TPL-Eager or TPL-Lazy perform best.

4.6 Conclusions

In this chapter we introduced the *EntryHyperplane* domination decision criterion which can be used to completely and correctly detect $Dom(A, B, R)$, with the restriction that B has to be a point. Additionally we developed a plane sweep algorithm which can be utilized to exactly compute the domination count $DC(\mathcal{O}, b, R)$ for the two dimensional case. Both of these techniques are then incorporated into the novel application reverse nearest neighbor ranking. The RNN ranking is formalized and an original solution is proposed. Our solution extends existing methods for RNN query processing in the following important aspects. First, we showed how the techniques for domination count estimation can be utilized for this problem and second we proposed an incremental algorithm for the RNN ranking problem. Our experimental evaluation confirms that our new solution outperforms existing methods adapted for the new problem significantly in terms of query execution times. Additionally we could show that incorporating the concept of partial domination for improving the domination count estimation yields another performance boost for this application.

Chapter 5

The CornerBased Domination Decision Criterion

In this chapter, we introduce the *CornerBased* domination decision criterion which can be used for detecting the domination relation $Dom(A, B, R)$, where all three objects A, B and R may be rectangular approximations.

As an application we formalize the novel concept of Constrained Reverse k-Nearest Neighbor (CRkNN) search on mobile objects (clients) performed at a central server. The CRkNN query computes for a given query object q the set RkNN(q) of objects having q as one of their k-nearest neighbors, iff the result set exceeds a specific threshold m, i.e. $|RkNN(q)| \geq m$. Otherwise, the query reports an empty result. In our setting, the positions of the query object and database objects are approximated by minimal bounding rectangles (safe regions) that depend on the last reported location of the object, as well as on the time that has been passed since the object reported its recent exact location. We propose an approach that minimizes the amount of communication between clients and central server by using the approximation of the positions to identify true hits and true drops. We present a multi-step filter/refinement framework that uses a novel refinement heuristic to minimize the number of objects that are required to provide their exact location. This heuristic is based on the *CornerBased* domination decision criterion and vastly reduces the required amount of network traffic compared to traditional refinement heuristics used for RkNN search. Our framework is applicable to both *mono-chromatic* and *bi-chromatic* CRkNN queries. Our solution does not assume any preprocessing steps which makes it applicable for dynamic environments where updates of the database frequently occur. Experiments show that our approach considerably reduces the communication load compared to existing approaches designed for traditional reverse nearest neighbor search in static data.

Parts of this chapter have been published in [62].The remainder is organized as follows: In Section 5.1 we introduce the *CornerBased* domination decision criterion and show how it can be utilized on order to detect $Dom(A, B, R)$. Afterwards we formalized the problem of CRkNN query processing in spatio-temporal data implementing a client-server setting and review related work in Section 5.2. Section 5.3 provides the details of our novel multi-

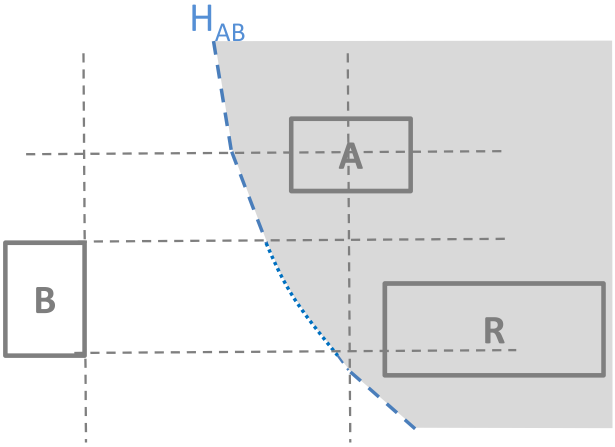


Figure 5.1: Geometric interpretation of Dom(A, B, R)

step CRkNN query processor and describes the complete query algorithm. A comparative experimental evaluation is presented in Section 5.4. Section 5.5 concludes the chapter.

5.1 The Criterion

A major restriction of the *EntryHyperplane* domination decision criterion is that for detecting $Dom(A, B, R)$, B has to be a point and not a spatial approximation with extension. In this section we want to show how this constraint can be removed.

5.1.1 Geometric Interpretation

The starting point for extending the previous approaches (Hyperplane and EntryHyperplane Criterion) is to consider the geometric interpretation of the problem. Therefore we need to identify the construct H_{AB} (Note that for detecting $Dom(A, B, R)$ we need to check if $R \in H_{AB}(A)$). In Figure 5.1 we illustrated H_{AB} for the two rectangles A and B. A naive solution for finding H_{AB} would be to materialize all possible hyperplanes H_{ab} defined by all pairs $(a, b) : a \in A, b \in B$. In order to guarantee no false dismissals, the corresponding half space $H_{AB}(A)$ has to be built by intersecting all half-spaces $H_{ab}(a)$. Obviously, the materialization of the hyperplanes of all possible pairs of points of A and B is not applicable. Rather, we only need to focus on the pairs of points which are responsible for constructing H_{AB}.

In fact, the half space $H_{ab}(a)$ built by points a and b has the property that all points $p \in H_{ab}(a)$ in this space are closer to a than to b, i.e. $dist(p, b) \geq dist(p, a)$ (cf. Figure 5.1). Now, let us again consider the object approximations A and B. The half space $H_{AB}(A)$ is represented by all points p that are definitely closer to each point in A than to each point in B, i.e.

$$\forall a \in A : \forall b \in B : dist(p, b) \geq dist(p, a). \tag{5.1}$$

In order get a conservative half space we can replace the distance between p and B by the minimum distance $MinDist(p, B)$ and the distance between p and A by the maximum distance $MaxDist(p, A)$. Consequently, we can define the half space by means of the following lemma:

Lemma 5.1. Let A and B be rectangles then the half space $H_{AB}(A)$ is defined by all points p for which the following equation holds:

$$MinDist(p, B) \geq MaxDist(p, A). \tag{5.2}$$

Proof. Since the half spaces defined by the two equations 5.1 and 5.2 are in fact identical, for the proof of the above lemma it suffices to show that each point p in the half space defined by Equation 5.2 is within the half space defined by Equation 5.1 and vice versa. Formally that $\forall a \in A : \forall b \in B : dist(p, b) \geq dist(p, a) \Leftrightarrow MinDist(p, B) \geq MaxDist(p, A)$

"$\Leftarrow$": Let p be any point, for which Equation 5.2 holds. Then, each point $b \in B$ has a larger distance to p than $MinDist(p, B)$, i.e.

$$\forall b \in B : dist(p, b) \geq MinDist(p, B).$$

Furthermore, each point $a \in A$ has a smaller distance to p than $MaxDist(p, A)$, i.e.

$$\forall a \in A : MaxDist(p, A) \geq dist(p, a).$$

Consequently, with Equation 5.2 we get the following equation:

$$\forall b \in B : dist(p, b) \geq MinDist(p, B) \geq MaxDist(p, A) \geq dist(p, a),$$

which is equal to Equation 5.1.

"$\Rightarrow$": Starting with Equation 5.1

$$\forall a \in A : \forall b \in B : dist(p, b) \geq dist(p, a)$$

we can deduce the special case

$$\Rightarrow \min_{b \in B} dist(p, b) \geq \min_{a \in A} dist(p, a)$$

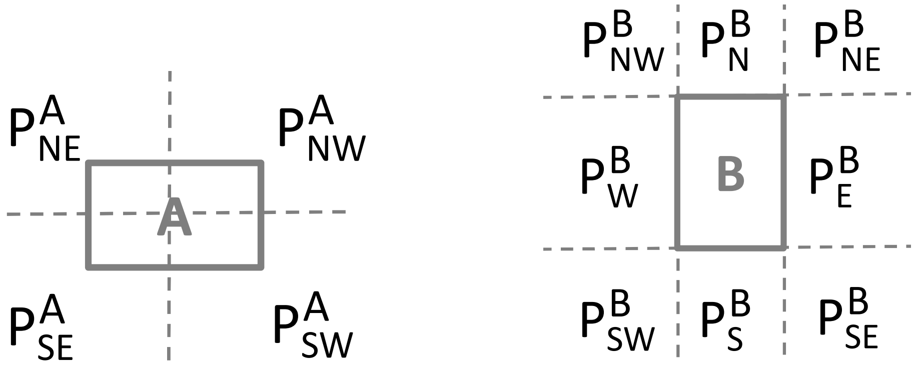
and substitute with the definition of the minimum and maximum distance

$$\Rightarrow MinDist(p, B) \geq MaxDist(p, A)$$

□

Based on Lemma 5.1 we can define the border of the hyper plane construct H_{AB} which is defined by all points p for which the following equation holds:

$$MinDist(p, B) = MaxDist(p, A). \tag{5.3}$$



(a) Partitioning of object A. (b) Partitioning of object B.

Figure 5.2: Partitioning of objects A and B for the 2d case.

In the remainder we will show how the pruning area can be constructed in consideration of the topology of the corresponding rectangles. The computation of the border of the half space $H_{AB}(A)$ defined by two rectangles in a two-dimensional space can be partially reduced to the basic case. In fact, we have to consider 36 cases, i.e. the space can be decomposed into $4 \cdot 9 = 36$ partitions having certain topological properties w.r.t. to both rectangles, as depicted in Figures 5.1 and 5.2. The rectangle B decomposes the space into 9 partitions along the rectangle margins (cf Figure 5.2(b)), 3 partitions per dimension. For each partition P we can determine a point or margin of the rectangle B that represents or contains the nearest point of B to any point in P. For example, consider the north-east partition P^B_{NE} built by B. The upper-right corner of B is the nearest point in B for all points in P^B_{NE}. Furthermore, the lower-right corner of B is the nearest point in B for all points in the south-east partition P^B_{SE}. Similar representatives can be found for the north-west and south-west partitions. The nearest point in B to any point in the east partition P^B_E is not a unique point and corresponds to the nearest point on the right rectangle margin. This also holds for the south, west and north partition P^B_S, P^B_W and P^B_N, respectively. Obviously, the partition which is defined by the rectangle B itself composes all points having a distance of zero to B.

The rectangle A decomposes the space into 4 additional partitions (cf Figure 5.2(a)), 2 partitions per dimension. Here, the center of A is used to define the partitioning which leads to the partitions P^A_{NE}, P^A_{SE}, P^A_{SW} and P^A_{NW}. In contrast to the partitions defined by B, here we are interested in partitions having the farthest point in A in common. For example, all points in the south-west partition P^A_{SW} built by A have the upper right point of A, i.e. the opposite corner of A w.r.t. P^A_{SW}, as their farthest point in A. Analogous considerations hold for the other three partitions.

Let us now consider the combined partitioning defined by the two rectangles A and B in order to determine H_{AB}. H_{AB} is composed of parts defined for each partition. In our example illustrated in Figure 5.1, H_{AB} within partition $P^B_{NE} \cap P^A_{SW}$ is defined by the upper-right corner of B and the upper-right corner of A. In contrast the partition $P^B_{NE} \cap P^A_{NW}$

defines H_{AB} by means of the upper-right corner of B and the lower-right corner of A. Each of the above parts can be represented by a single line segment. We achieve a more complex structure when considering H_{AB} within partition $P_E^B \cap P_{SW}^A$. Since, this partition has not a common nearest point in B, i.e. the nearest point in B depends on the location of the point $p \in H_{AB} \cap P_E^B \cap P_{SW}^A$, the geometry of the line segment is not linear anymore. The resulting structure of H_{AB} is now rather complex, persisting of several line segments and even non linear parts (see the narrow dotted blue line in Figure 5.1). The materialization of H_{AB} in order to test whether an object R lies completely within $H_{AB}(A)$ is thus not trivial and could be computationally expensive.

5.1.2 Efficient Computation

Since the geometrical interpretation based on the half space as described above is effective but the materialization of the pruning area is expensive, we have to find a method to check whether a point or rectangle is completely within $H_{AB}(A)$ without requiring to materialize H_{AB}. Note that in figure 5.1 it is sufficient to only test whether the corners of R are within $H_{AB}(A)$, in order to decide if R as a whole is within $H_{AB}(A)$. In the following, we will show that this approach can be generalized.

First, we utilize the fact that $H_{AB}(A)$ always has a convex structure which is shown by the following two lemmas:

Lemma 5.2. Let $P_1, \ldots, P_n \subset \mathbb{R}^d$ be convex areas. Then the intersection $P = P_1 \bigcap \ldots \bigcap P_n$ is also convex.

Proof. Let $p_1, p_2 \in P$, then p_1 and p_2 are within every set P_j. Since the sets P_j are convex, the whole segment $[p_1, p_2]$ lies within each set P_j. Since this segment lies within each set P_j, it also is part of the intersection of these sets, i.e. $[p_1, p_2] \subset P$. Thus the intersection P of finite number of convex sets is also convex. □

Lemma 5.3. The half space $H_{AB}(A)$ defined by the two rectangles A and B is convex.

Proof. The half space $H_{ab}(a)$ defined by any two points $a \in A$, $b \in B$ is a half-plane, which is a convex set. The resulting area when intersecting the half-planes of all pairs (a, b) is again convex as a consequence of Lemma 5.2. □

Now, we can use the convexity property of $H_{AB}(A)$ in order to check whether a point or rectangle is completely within $H_{AB}(A)$ in an efficient way using the following lemma.

Lemma 5.4. Let R_v be the set of vertices of the rectangle R. In order to check weather $Dom(A, B, R)$ holds it suffices to show that for all vertices $r_v \in R_v$ of R the following criterion holds:

$$MaxDist(r_v, A) \leq MinDist(r_v, B)$$

Proof. According to Lemma 5.3 B and A form the convex half space $H_{AB}(A)$. Therefore if R is not completely within $H_{AB}(A)$, at least one corner r_v of R must be outside $H_{AB}(A)$. Thus the following inequality $MinDist(r_v, B) < MaxDist(r_v, A)$ holds. □

As a consequence of Lemma 5.4 the test if a rectangular object R lies within$H_{AB}(A)$ formed by two rectangular regions A and B can be reduced to the *MinMaxDist* domination decision criterion of the corners of R. This domination criterion is complete and correct as it is based on the exact pruning area defined by A and B. In particular, it is different from DDC$_{MM}$ which applies the domination test only for the complete rectangles rather than to the corner points and, thus, is significantly less selective.

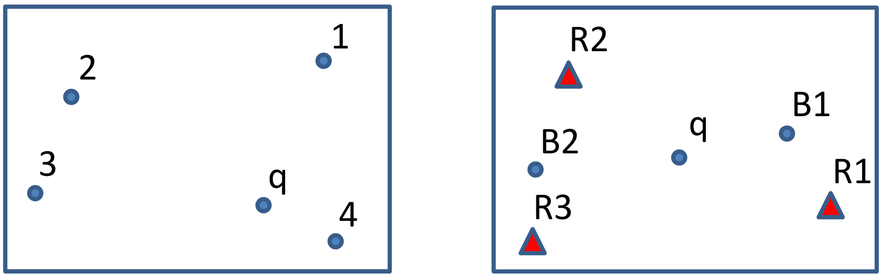
Definition 5.1 (Corner-Based domination decision criterion)**.** Let $A, B, R \in \mathbb{R}^d$ be rectangles in $\mathbb{R}^d$ and R_v be the the set of vertices (corners) of R. The *CornerBased* domination decision criterion is defined as

$$\mathrm{DDC}_{CB}(A, B, R) \Leftrightarrow \forall r_v \in R_v : MinDist(r_v, B) < MaxDist(r_v, A).$$

Regarding the efficiency the *CornerBased* domination decision criterion requires to perform a distance computation for each vertex of R in the worst case. Thus the runtime is $O(2^d)$. However for spatial applications with a small dimensional space this criterion is suited very well.

5.2 Application: Constrained Reverse Nearest Neighbor Search on Mobile Objects

In contrast to the reverse k-nearest neighbor search problem on static data, considerably less work has been done so far to support RkNN queries on mobile objects that may not be indexed by a point access method. The prevalence of inexpensive and very small Global-Positioning-Devices (GPS) gives rise to new spatio-temporal database applications. With these advances it is e.g. possible to track pupils by equipping them with such a GPS-device or even animals by attaching a GPS-device to their ear or under their skin. However, such very small GPS-devices posses only a very limited power supply, and the replacement of a power supply of such a GPS-device can be very expensive. As an example, think of a wildlife sanctuary, where many animals are equipped with small GPS-devices to observe and protect them. The GPS signal of the animals can be used to alert the ranger, when a very rare species is in immediate danger of being attacked by a predator, such as a tiger. The ranger may then intervene by chasing off the tiger. However, the act of attaching a GPS-device to an animal or replacing its power source is very stressful and dangerous for both the animal and the rangers. Usually, the animal has to be tranquilized in order to allow the veterinary a safe approach. Thus, the power of such miniature GPS-devices is a very precious resource. In order to preserve this source, the miniature GPS-device should only submit its exact position if necessary. As a consequence, in such an application scenario, query processing must account for the fact that any position poll necessary to answer the query is very costly, or in other words, each single position poll that is saved is valuable.



(a) Mono-chromatic case.

(b) Bi-chromatic case.

Figure 5.3: R1NN examples.

Many applications also only require to know if the number of RkNNs of a query object q exceeds a given threshold m. If q has less RkNNs than m, no results need to be reported. Only if the number of RkNNs of the query is at least m, all RkNNs need to be known. This type of query is called Constrained Reverse k-Nearest Neighbor (CRkNN) query. As an example application for a CRkNN query, consider lions in a wildlife sanctuary. Since lions generally hunt in packs, it is safe to assume that the query animal (or a tourist) q is not in danger if less than $m = 3$ lions have q as their nearest prey. If however the number of reverse nearest lions of the query exceeds the threshold $m = 3$, then the lions need to be chased off (or the tourist has to be warned).

To track and observe large numbers of continuously moving objects, their last submitted positions are stored in a database. The conventional assumption, that data remains constant unless it is explicitly modified, no longer holds when considering mobile objects. In our examples above, the position of an animal may be given by an exact location at the time slot the animal submitted its current position. After that, its position is conservatively approximated by a minimal bounding box that contains all possible positions the lion may have reached since its last location update and that usually grows over time.

We consider the computation of CRkNN queries in this setting for both the mono-chromatic as well as the bi-chromatic case. The difference of both cases is that in the mono-chromatic case we have only one set of objects from which the query and the results are drawn, whereas in the bi-chromatic case, we have two sets, a set of potential query objects (that are not considered as results when a query is processed) and a set from which the results are drawn. Figure 5.3 illustrates the concept of CRkNN queries in both cases. The mono-chromatic CR1NN query (cf. Figure 5.3(a)) for point q returns point 1 and 4 if $m \in \{1, 2\}$ or nothing if $m \geq 3$. Points 2 and 3 are not returned for any choice of m, because they do not find q as their 1-nearest neighbor. In the bi-chromatic case, two object sets $\mathcal{D}_{red}$ (red objects) and $\mathcal{D}_{blue}$ (blue objects) are considered. The bi-chromatic CRkNN query returns all elements of $\mathcal{D}_{red}$ that have the query point as on of their k-nearest neighbors if all other red objects are ignored. Figure 5.3(b) shows the bi-chromatic CR1NN query for a set of lions (red objects $R1$, $R2$, and $R3$) and a set of potential prey (blue objects $B1$, $B2$, and q). The query object q (from the blue object set) could be a young elephant that is not yet able to defend itself. In this example, the bi-chromatic CR1NN query yields no results for any value of m, because each lion observes another animal as nearest neighbor.

A straightforward solution for computing CRkNN queries is to check for each point whether it has a given query point as one of its k-nearest neighbors. If more than m objects do, the set is output as result, otherwise the result is empty. However, this approach lacks on a computational point of view when the number of points is large. Even more important in this setting is that each client is required to send its exact location at least once and, thus, the drain of the clients' power sources is very large.

5.2.1 Problem Formalization

Client-Server Scenario

In the following, we assume that $\mathcal{D}$ is a database of n objects (clients) moving continuously within a 2-dimensional Euclidean space and *dist* is the Euclidean distance[1] on the objects in $\mathcal{D}$. In addition, we assume that the objects (clients) are connected with a central server via a wireless network and can send their exact positions when requested from the server.

At server side the position of each object o is approximated by a two-dimensional axis aligned rectangle *o.mbr* that minimally bounds the possible positions of o. The objects send their exact positions to the server only if necessary. The exact position of an object o will not be updated at server side as long as

- the exact object position *o.pos* is within the region *o.mbr*.
- the query can be answered based on the information of the object positions that is currently available at the server.

Here, we will be interested in performing queries among the clients at the server.

CRkNN Query

The problem of a *constrained RkNN* (CRkNN) query is to report for a given query object q the result of a RkNN query if and only if the result size exceeds a specific threshold m, formally

$$CRkNN(q) = \begin{cases} RkNN(q) & \text{, if } Card(RkNN(q) \geq m \\ \emptyset & \text{, else.} \end{cases}$$

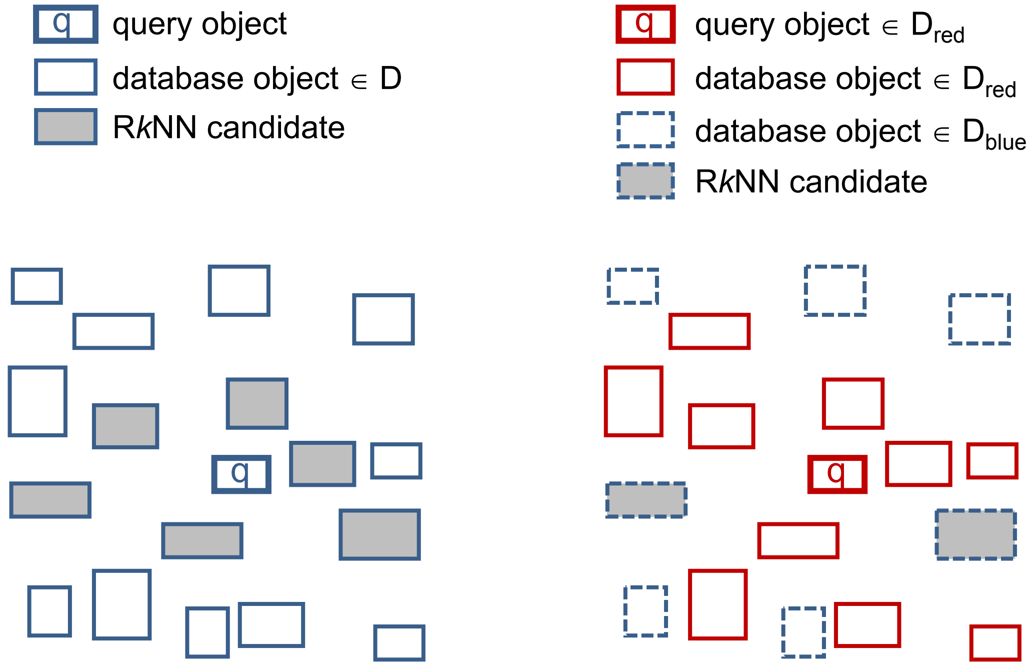
Note that the case where $m = 1$ corresponds to a traditional, non-constrained RkNN query.

5.2.2 Related Work

Although our problem definition of *constrained RkNN* is new, it is obviously closely related to common RkNN queries. Techniques for RkNN query processing in static data have been discussed in Section 2.4. For moving objects data there are relatively few works which address the RkNN problem. Most of the existing work in this direction focus on problem

---

[1] The concepts described here can also be extended to any L_p-norm and any dimension $d > 2$



(a) $Card(R1NN(q)) = 6$ (monochromatic).

(b) $Card(R1NN(q)) = 2$ (bichromatic).

Figure 5.4: R1NN selectivity estimation based on position approximations.

of continuous reverse nearest neighbour monitoring [170, 90, 168], where the task is to find for a given query point q the reverse k-nearest neighbours for each point of time. The main focus lies always on reducing IO and CPU operations (since the position of each object is known at each point of time) and the query has to be a single point. This is of course different from our scenario since we want to reduce communication cost and the query can be a moving object too. In [23] the authors focus on snapshot RkNN queries on moving objects. But again the query has to be a single point and the positions of the objects in the database have to be known at each point of time.

5.3 CRkNN Query Processing

In the following sections, we present our framework for answering CRkNN queries on objects initially approximated by minimal bounding rectangles.

5.3.1 General Idea for Answering CRkNN Queries

The main objective of our approach is to keep the communication cost between the clients and the server as low as possible while answering CRkNN queries for a given query object q, and specified values for k and m that may vary from query to query. For this reason, we try to avoid unnecessary position updates at the server side. The communication cost of each position update is assumed to be very expensive, since it composes the cost required to build up the communication channel between the server and the client, and the cost for transferring the position information from the client to the server. The problem is that we might update object positions when issuing CRkNN queries in order to get exact results. However, the CRkNN query offers a potential for a lazy update strategy. Whenever a CRkNN query is issued, first, we estimate conservatively the selectivity of the query only based on the information available at the server, i.e. in consideration of the object approximations. An example illustrating this estimation task for the mono-chromatic and the bi-chromatic case is depicted in Figure 5.4. Detailed strategies for estimating the selectivity, i.e. finding the RkNN candidates, will be discussed in the next section. Only if the estimated size of the result set exceeds a specified threshold m, then the server requests exact position updates from the clients. In our example, m must be smaller than 6 in the mono-chromatic case (cf. Figure 5.4(a)) and smaller than 2 in the bi-chromatic case (cf. Figure 5.4(b)). Thereby, only those position updates are requested that are necessary to compute the exact result. The determination of those objects (including the query object) that definitely have to transmit their exact positions to the server is challenging. In summary, there are two problems to be solved:

- First, we have to compute a conservative but accurate estimation of the query selectivity.

- Secondly, we have to find a possibly small set of objects whose refined positions suffice to compute the correct query result.

Obviously, the higher the accuracy of the position information of the query object and the database objects at sever are, the better is the query selectivity estimation. However, since our main focus is to keep the position updates as low as possible, we first estimate the query selectivity without any position update. For the query selectivity estimation we have to compute RkNN candidates in consideration of the minimal bounding rectangles associated with the query object and the database objects. For a clear presentation, we focus on the mono-chromatic case but all concepts can be directly applied to the bi-chromatic case.

5.3.2 Pruning Strategies

First, we will introduce pruning strategies which are applicable for extended query objects and are used in a filter step to identify potential candidates and, thus, to estimate the selectivity of the query. Thereby, the main focus is to cut down the number of CRkNN candidates to achieve a good conservative estimation of the selectivity of an RkNN query and to identify a possibly small number of objects for which the server has to request the exact positions.

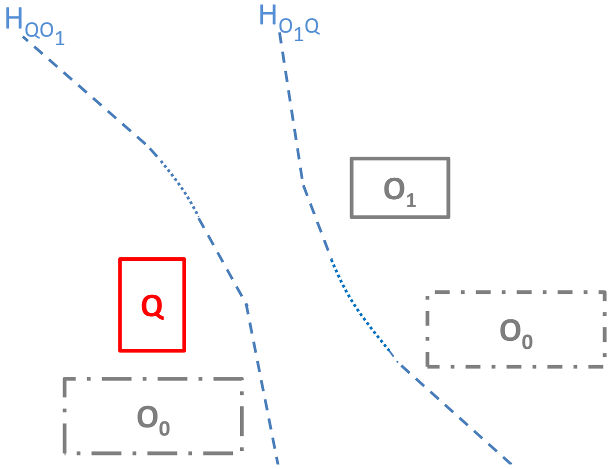


Figure 5.5: Conservative and progressive pruning.

Conservative Pruning

For excluding objects from further consideration during query processing we can straightforwardly use the *CornerBased* domination decision criterion. If for two objects (approximated by rectangles) $O_0, O_1 \in \mathcal{D}$ from the database and a query object Q it holds that $\mathrm{DDC}_{CB}(O_1, Q, O_0)$, then O_0 can be pruned since it can not be RNN of Q (cf. Figure 5.5 assuming O_0 is on the right-hand side). Of course we can extend this pruning to values of k larger than 1. Therefore we basically perform a naive estimation of the domination count of O_0 by counting the number of objects in the database which dominate Q according to O_0. Formally we can prune an object O_0 if:

$$|O_i \in \mathcal{D} \setminus O_0 : \mathrm{DDC}_{CB}(O_i, Q, O_0)| \geq k$$

Progressive Pruning

Up to now, we presented conservative pruning technique in order to prune candidates that can be excluded from the query result. Now, we present additional pruning strategies called *progressive pruning* that find true hits without any refinement of candidates. Thereby, we apply similar geometrical properties as used for the conservative pruning. For a given query rectangle Q and a database rectangle O_1 the progressive pruning area $(H_{O_1Q}(Q))$ defines all points that are definitely closer to all points in Q than to all points in O_1. An example is illustrated in Figure 5.5(assuming O_0 is on the left-hand side). In contrast to the margin of the conservative pruning area (H_{O_1Q}), the margin of the progressive pruning area is defined by H_{QO_1}. Hence, if a point or rectangle (O_0 in this example) completely lies within this area, it can be concluded that it is closer to any point in Q than to O_1. Detecting this relation can be achieved by using the Corner-Based decision criterion by switching parameters $(\mathrm{DDC}_{CB}(Q, O_i, O_0))$. For a RNN query, if a point or rectangle completely lies within the progressive pruning area of each database object (except itself), it can be immediately reported as RNN(q) result. Again, this technique is extendible to the RkNN-problem where an object O_0 can be progressively pruned if it lies completely within $|\mathcal{D}|-k-1$ progressive pruning areas (defined by Q and $|\mathcal{D}|-k-1$ database objects).

5.3.3 Refinement Strategies

After applying our pruning strategies, there may still candidates left. For these candidates, the decision whether they are part of the result cannot be made by using only information stored on the server. In this case, one of the objects of the database has to be refined, i.e. one of the objects is required to transmit its exact location to the server. In this section we propose two strategies that aim to minimize the total number of refinements. This is important since we follow our overall goal in minimizing the number of messages sent between clients and server.

Minimal Mindist Heuristics

The idea of refining the object of the database, that has the smallest Mindist to the query object q has been proposed in [152]. This heuristic has also been adapted in [4]. The general idea is, that objects closest to q are likely to be true hits and also generate pruning areas close to q that potentially prune other objects that are further apart. Thus, for each object approximation O, we compute the minimal distance $MinDist(q,O)$ to q, and refine object

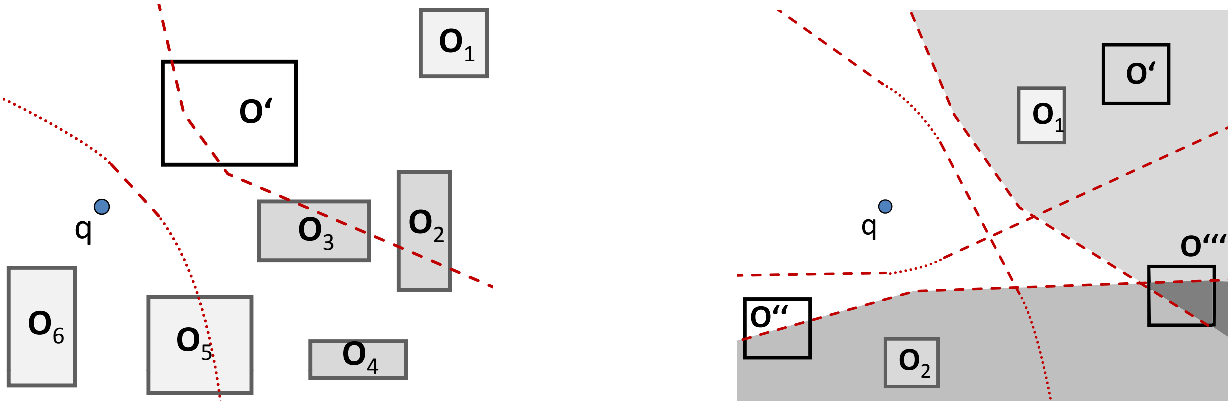
$$argMin_{O\in DB}(MinDist(O,q)).$$

Maximum Pruning Potential Heuristics

The key of this idea is to pick the object that has the most influence on all other candidates as next object to be refined (exat position is polled by the server). In particular, that object for which the likelihood, that due to its refinement, other candidates can be pruned, is maximized should be chosen.

To evaluate this influence, each object O is associated a pruning potential. The pruning potential of an object is composed of two components, the Mutual-Pruning Potential, $MPP(O)$, and the Self-Pruning Potential, $SPP(O)$.

The Mutual-Pruning Potential of an object O estimates the number of candidates that can possibly be pruned by the exact position o of O, but cannot be pruned by means of the approximation of O only. An example of this estimation for a $CR1NN$ query is given in Figure 5.6(a). In this example, we want to determine the Mutual-Pruning Potential of O', $MPP(O')$. Object O_1 does not affect $MPP(O')$, because O_1 can already be pruned based on the approximation of O'. O_6 does not contribute to $MPP(O')$ either, because regardless of the exact position of $o \in O'$, O_6 cannot be conservatively pruned by O'. The reason for this is that O_6 cannot be conservatively pruned by any line that is contained in the region between the conservative and the progressive approximation of the Voronoi lines between q and any $o' \in O'$ regardless of the exact position of O_6. The same applies for O_5. Even in the best case, where $o' \in O'$ is very close to q, O_5 cannot be completely pruned by



(a) Mutual Pruning. (b) Self Pruning.

Figure 5.6: Different pruning potentials.

o'. Let us note that it is possible that the Voronoi line between q and o' intersects O_5, and by means of partial pruning, the prune count of O_5 can still be increased in combination with Voronoi lines of other objects. Thus O_5 increments $MPP(O)$ by one. For objects O_2, O_3 and O_4 there exist possible positions $o' \in O'$, for which they can be completely pruned. Thus, the Mutual-Pruning Potential of O' is incremented by one for each of these objects and finally we get $MPP(O') = 4$. In the bi-chromatic case, the following additional constraints apply:

- The object O for which the Mutual-Pruning Potential is computed must be a member of set $\mathcal{D}_{blue}$ (the *blue* objects), because objects of set $\mathcal{D}_{red}$ (the *red* objects) do not interfere with other objects, and thus cannot prune other candidates.

- The objects (O_1,...,O_6 in the example) that increase the Mutual-Pruning Potential of O must be *red*, since the result of a bi-chromatic CRkNN query may only contain *red* objects. Therefore only *red* objects can be candidates.

Thus, *red* objects do not have a Mutual-Pruning Potential and *blue* objects ignore other *blue* objects in the computation of their Mutual-Pruning Potential.

The Self-Pruning Potential of an object O estimates the number of objects that can possibly conservatively prune the exact position of $o \in O$, but cannot conservatively prune the approximation O. An example for this situation is depicted in Figure 5.6(b). Here, we want to determine the Self-Pruning Potential $SPP(O')$ of O'. Object O_1 does not increase the Self-Pruning Potential of O', because the approximation of O' can already be pruned completely by O_1. O_2 does not increase the Self-Pruning Potential of O' either, because regardless of the exact position of $e' \in O'$, o' cannot be pruned. Thus, $SPP(O') = 0$. Now, consider the Self-Pruning Potential of O''. Although O_2 cannot prune the approximation O'', O_2 may be able to prune $o'' \in O''$ if it falls into the pruning region of O_2, denoted by the shaded area. Therefore, O_2 increases the $SPP(O'')$ by one. The situation of O''' is even more interesting, because here, both O_1 and O_2 contribute to the Self-Pruning Potential of O'''. Again, in the bi-chromatic case additional constraints apply including:

- The object O for which the Self-Pruning Potential is computed must be a member of set $\mathcal{D}_{red}$ (the set of *red* objects), because the result of a bichromatic (C)RkNN query may only contain *red* objects, and thus, only *red* objects may be candidates.
- The objects (O_1 and O_2 in the example) that increase the SPP of candidates must be *blue*, because only members of the blue set can prune other objects.

Thus, *blue* objects do not have a Self-Pruning Potential, and *red* objects ignore other *red* objects in the computation of their Self-Pruning Potential.

After computing $MPP(O)$ and $SPP(O)$ the object with the highest sum of Mutual-Pruning Potential and Self-Pruning Potential

$$argMax_{O \in DB}(MPP(O) + SPP(O))$$

is chosen to be refined next.

As seen in the examples, in order to compute $MPP(O)$ and $SPP(O)$ for an object O, we need to find objects that are not completely contained in either the pruning region or in the true-hit region of other objects. Due to the convexity of both the true-hit and the pruning region of an object O, we can do this efficiently by testing corners of MBRs only (see above).

Algorithm 4 CRkNN($Q, k, m, \mathcal{D}$)

```
$cands = O_i \in \mathcal{D} | \sharp\{O_j \in \mathcal{D} | \mathrm{DDC}_{CB}(O_j, Q, O_i)\} < k\}$
$result = \emptyset$
if $\sharp cands < m$ then
    return $\emptyset$
else
    $refine(Q)$
    $cands = \{O_i \in cands | \sharp\{O_j \in \mathcal{D} | \mathrm{DDC}_{CB}(O_j, Q, O_i)\} < k\}$
    while $\sharp result + \sharp cands > m$ && $\sharp cands > 0$ do
        $refine(argMax_{O \in \mathcal{D}}$(refinement heuristic)
        $cands = \{O_i \in cands | \sharp\{O_j \in \mathcal{D} | \mathrm{DDC}_{CB}(O_j, Q, O_i)\} < k\}$
        $result = \{O_i \in cands | \sharp\{O_j \in \mathcal{D} | \mathrm{DDC}_{MM}(Q, O_j, O_i)\} > \sharp\mathcal{D} - k\}$
    end while
    if $\sharp result > m$ then
        return $result$
    else
        return $\emptyset$
    end if
end if
```

5.3.4 Algorithm

In this section we present our algorithm for processing CRkNN queries on approximated objects. An abstract outline of the algorithm is shown in Algorithm 4. In the first step, the algorithm applies pruning based on extended regions utilizing the information that is available on the server. If the number of candidates returned in this step is less than m the algorithm terminates with the empty set as result, since the number of results can not possibly exceed m. Otherwise, if the number of candidates is at least m, then the query object Q is refined, and thus required to send its exact position to the server. Then the candidate and result lists are updated according to the new information about the exact position of Q using the various pruning strategies described above. This is repeated while there are candidates left to be refined. If at any time, the possible number of results, i.e. the number of current hits ($result$) plus the number of remaining candidates $cands$ becomes less than m, the algorithm terminates again with the empty set as result. Once no more candidates are left, the results are returned if their number exceeds m.

5.4 Experimental Evaluation

In the following, we outline the experimental evaluation. Since all proposed techniques aim at reducing the number of refined objects, the number of refinements is the main variable to be measured. In this regard we will show:

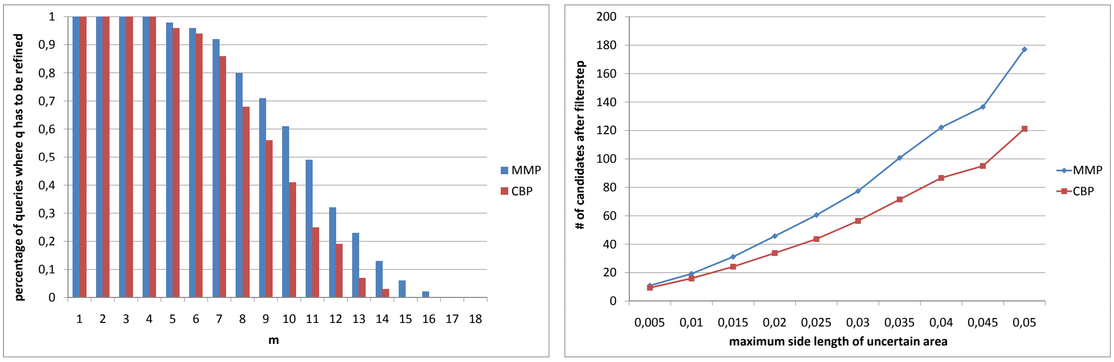
- A comparison of the efficiency gain of DDC$_{CB}$ over DDC$_{MM}$. Therefore we implemented two different algorithms. One is the algorithm described in Algorithm 4, is based on DDC$_{CB}$ and named *Corner-Based-Pruning (CBP)*. The second one is an adaption of Algorithm 4, where we replaced all occurrences of DDC$_{CB}$ through DDC$_{MM}$. This algorithm is called *MinMaxDist-Pruning (MMP)*.
- A comparison of the two proposed refinement strategies *Minimal Mindist-Heuristics (MMH) and Maximum Pruning Potential Heuristics (MPPH)*.
- The impact of the constraint parameter m.

The experiments will be performed under various different parameter settings as well as on different datasets and in both monochromatic and bichromatic environment. Default values for parameters are $k = 5$, $m = 1$, $|\mathcal{D}| = 10k$, $|\mathcal{D}_{red}| = 10k$ and $|\mathcal{D}_{blue}| = 10k$.

5.4.1 Datasets and Parameters

For the evaluation of the different approaches we used two different 2D point datasets:

- Synthetic dataset with uniform distribution (2D-Uniform)
- The Twin Astrographic Catalog Version 2 [175]



(a) Relative number of refinements of q

(b) Number of candidates after the first filter step

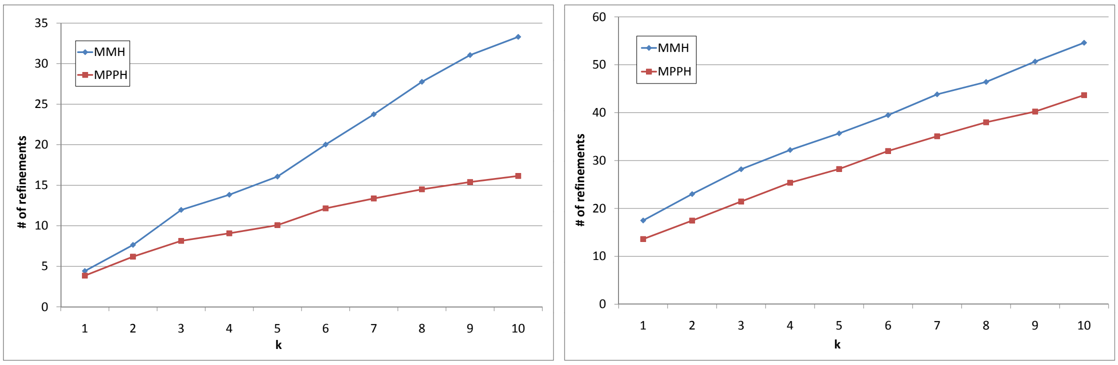
Figure 5.7: MMP vs. CBP on uniform dataset

Both datasets were normalized to have a feature value in [0,1]. Each point is equivalent to the exact position of one object. The uncertain area of each object o is constructed by choosing a random rectangle that covers o with a random side length in $[0, maxsidelength]$. Queries as well as database objects were picked from one of the datasets for each experiment.

5.4.2 Filter Step

In our first experiment we evaluate the pruning power of our pruning strategy CBP in comparison to the basic pruning strategy MMP. First, we investigate the cost required to refine the query object for varying threshold parameter m in terms of the average number of query object refinements. Note, that in the case where refining the query object is not required, we have no refinement costs at all, i.e. no database object has to be refined. The results are depicted in Figure 5.7(a). For very small m ($m < 5$) our advanced approach CBP has no improvement in comparison to the simple MMP approach because for a small m almost in all cases the result set $|RkNN(q)|$ exceeds m and, thus, the query object has to be refined. For larger settings of the parameter m, indeed we can save refinements of the query object and, thus, can save the computation of the CRkNN(q) result. With increasing the parameter m, we can observe that our geometrical pruning strategy increasingly outperforms MMP. In particular, for $m = 11$ the CBP is two times more selective than the MMP approach.

Furthermore, we investigated the pruning power of both pruning strategies in consideration of the number of result candidates reported by the first filter (without any refinement). Figure 5.7(b) shows the number of candidates generated in the first filter for varying sizes of the rectangles used to approximate the object positions in terms of the side length of the rectangles. It is obvious that larger rectangles lead to a lower pruning power with both competing approaches. However, the CBP pruning produces about $1/3^{rd}$ less candidates compared to MMP.



(a) MMH vs. MPPH on uniform data (b) MMH vs. MPPH on tac dataset

Figure 5.8: MMH vs. MPPH on uniform and tac dataset

5.4.3 Refinement Strategy

The next set of experiments concerns the effectivity of the proposed refinement strategies. As a baseline, we use the *Minimal Mindist Heuristics* (MMH) (c.f. 5.3.3) and compare it to our proposed *Maximum Pruning Potential Heuristics* (c.f. 5.3.3). We measure the total number of refinements, that is, the number of objects that are required to submit their exact position. Figure 5.8(a) shows the result of our evaluation with respect to k on uniform dataset. It can be observed that the $MPPH$ shows only a small improvement over the the MMH for small values of k, but becomes significant outperforms MMH as k increases. In contrast, the same evaluation is shown in Figure 5.8(b) for the tac dataset. It can be observed that the total number of refinements is higher on the tac dataset. This can be explained by the clustered nature of the tac dataset. Objects are closer to each other than in the uniform dataset, while the max side length of an uncertain area is the same, resulting in more overlap of the uncertain regions. It can also be observed that the improvement of $MPPH$ is relatively large for small values of k, increases rather slowly as k increases.

5.4.4 Constraint Parameter m

As the parameter m constrains the minimum number of RkNN results, which are of interest to the user, it can be used to stop the refinement process at an early stage. Thus, increasing m results in a smaller number of refinements. Figure 5.9 again shows the number of necessary refinements (using $MPPH$) with respect to k, but with different values of m. It can be seen, that for the same number of k, the number of refinements can be enormously reduced by a bigger m. Keep in mind that the case where $m = 1$ corresponds to a traditional, non-constrained RkNN query.

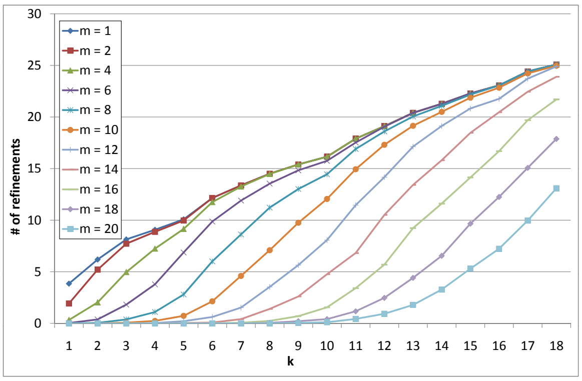
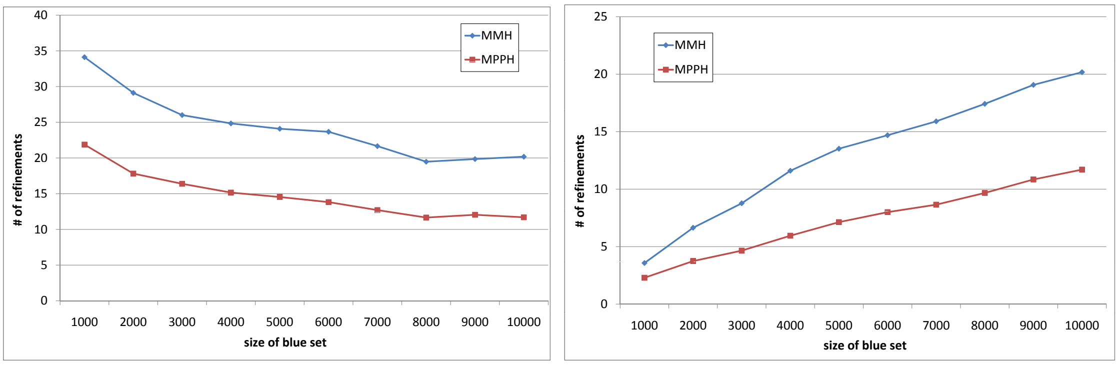


Figure 5.9: Impact of parameter m on the number of refinements



(a) Varying $|\mathcal{D}_{red}|$ (b) Varying $|\mathcal{D}_{blue}|$

Figure 5.10: MMH vs. MPPH with varying size of the two sets

5.4.5 Bichromatic Case

In the bichromatic case the interesting variable is the propotion of $|\mathcal{D}_{red}|$ to $|\mathcal{D}_{blue}|$. Figure 5.10(a) shows the number of refinements dependent on the size of $\mathcal{D}_{red}$, where the size $\mathcal{D}_{blue}$ is fixed to 10000. Figure 5.10(b) shows the opposite case, where Card($\mathcal{D}_{red}$) is set to 10000 and Card($\mathcal{D}_{blue}$) varies from 1000 to 10000. In both cases MPPH outperforms MMH, using up to 50% fewer refinements.

5.5 Conclusions

In this chapter we introduced the *CornerBased* domination decision criterion which is able to detect $Dom(A, B, R)$ when all three objects are given by rectangular approximations. The only existing domination decision criterion which is able to detect this relation is the MinMax domination decision criterion. To show the effectivity of DDC$_{MM}$ compared to DDC$_{MM}$ we additionally formalized a novel server-side reverse k-nearest neighbor problem for moving clients, the constrained reverse k-nearest neighbor (CRkNN) query. In a client/server scenario, the bottleneck for the resources is typically not the query execution time like I/O or CPU costs at clients or at the server. Rather, the communication load needs to be minimized because the most precious resource is the power supply of the clients. Especially in applications, where miniature GPS-devices with low power resources are used and the exchange of single devices is very complex, the goal of query processing is to save even single messages between clients and the server. While existing methods for the traditional RkNN problem are not directly designed to optimize the communication load but rather the query execution time, we propose an original solution for CRkNN queries in such a client/server application. In fact, we propose several new pruning strategies in order to keep the number of candidates as low as possible using only approximative information on the location of the clients. Our experimental evaluation confirms that our novel solution is superior to existing approaches adapted to the CRkNN problem in terms of communication costs.

Chapter 6

The Trigonometric Domination Decision Criterion

With the *CornerBased* domination decision criterion it is possible to detect $Dom(A, B, R)$ for three rectangular objects complete and correct. However this criterion can not be evaluated efficiently from a computational point of view when the number of dimensions d of the space containing the rectangles increases. The main problem is that for each corner of the rectangle R we have to perform a MinDist and a MaxDist computation. Since the number of corners of a multidimensional rectangle in d-dimensional space is 2^d the runtime of this criterion increases exponential in the number of dimensions.

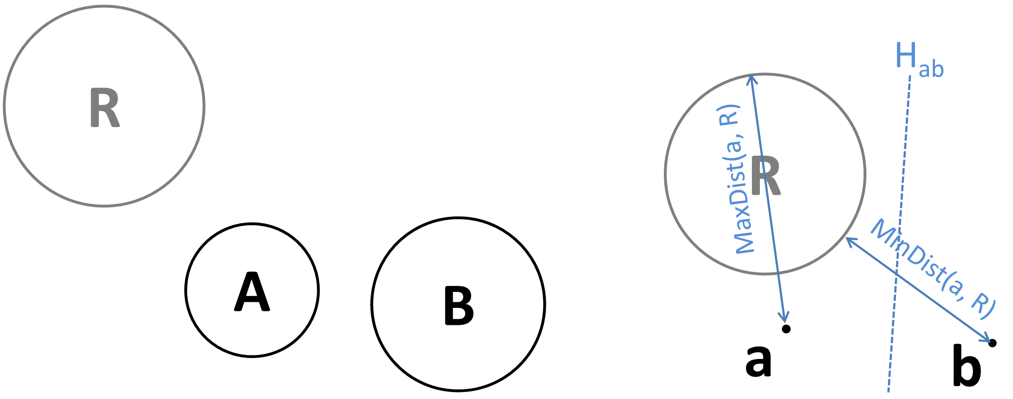
In this chapter we will introduce the *Trigonometric* domination decision criterion which is able remove this drawback. We achieve this goal by switching the shape of the approximation of the objects A, B and R from rectangles to spheres. After this little trick, it is possible to detect $Dom(A, B, R)$ through clever application of trigonometric functions.

To show the efficiency and effectivity of the proposed criterion we incorporated the technique into an all-(k)-nearest-neighbor (AkNN) query processing framework. AkNN is the task to determine the k-nearest neighbors for multiple query points simultaneously.

Parts of this chapter have been published in [58]. The rest of the chapter is organized as follows. In Section 6.1 we review how existing domination decision criteria can be adapted to spherical approximations and introduce the *Trigonometric* domination decision criterion. In Section 6.2 we discuss the problem of all-k-nearest neighbour queries, review related work and present basic solutions. Afterwards we show how to employ the *Trigonometric* domination decision criterion for AkNN query processing in Section 6.3. A performance evaluation is presented in Section 6.4. The chapter concludes with a summary in Section 6.5.

6.1 Domination Decision Criteria for Spheres

The main problem of DDC_{CB} is the computational complexity, which is exponential in the number of dimensions d of the space $\mathbb{R}^d$ in which the objects are defined. In this section we



(a) Does $Dom(A, B, R)$ hold?

(b) Hyperplane and MinMaxDist domination for spheres.

Figure 6.1: Domination on spheres

will introduce a decision criterion which is able to detect $Dom(A, B, R)$ for the case where objects A, B and R are given by multidimensional spheres (i.e. a multidimensional vector $c \in \mathbb{R}^d$ representing the center of the sphere and a single value $r \in \mathbb{R}$ representing the radius of the sphere) as illustrated in Figure 6.1(a). As a constraint the proposed technique is restricted to Euclidean distance (L_2-norm) which however is one of the most commonly used distance functions for similarity search.

6.1.1 Adopting existing criteria to spherical approximations

Before getting into the details of the Trigonometric domination decision criterion, we will first review how to adapt the *MinMaxDist* and the *Hyperplane* domination decision criterion to the case where the objects are approximated by spheres.

Therefore we will first define the minimum and maximum distance between two spheres.

Definition 6.1 (MaxDist and MinDist for Spheres)**.** Given two spheres B_i and B_j the *MaxDist* between B_i and B_j is defined as:

$$MaxDist(B_i, B_j) = dist(B_i.c, B_j.c) + B_i.r + B_j.r$$

The *MinDist* between B_i and B_j is:

$$MinDist(B_i, B_j) = dist(B_i.c, B_j.c) - B_i.r - B_j.r$$

where $B_x.c \in \mathbb{R}^d$ is the center and $B_x.r \in \mathbb{R}$ is the radius of sphere B_x and $dist()$ is the Euclidean distance function.

Given these definitions we can define the *MinMaxDist* domination decision criterion for spheres analogously to the the one for rectangles (see Definition 3.3).

Definition 6.2 (MinMaxDist Domination Decision Criterion for Spheres)**.** Given three spheres A, B and R the the MinMaxDist domination decision criterion is defined as

$$\mathrm{DDC}_{MM}(A, B, R) \Leftarrow MaxDist(A, R) < MinDist(B, R)$$

Though the *MinMaxDist* domination decision criterion for spheres is correct (proof is analogous to Proof 3.2.2), it is not complete. A counter example (showing that $\mathrm{DDC}_{MM}(A, B, R)$ for spheres is not complete) is illustrated in Figure 6.1(b). In this example $\mathrm{DDC}_{MM}(A, B, R)$ does obviously not hold, though $Dom(A, B, R)$ holds (This can be verified by the *Hyperplane* domination decision criterion for dpheres which is defined in the next definition).

The computation of the MinDist and the MaxDist on spheres can be performed in $O(d)$, since it only involves a distance computation between the two d-dimensional center vectors and two additions of the radii of the two spheres. Thus also $\mathrm{DDC}_{MM}(A, B, R)$ for spheres can be computed in $O(d)$.

Definition 6.3 (Hyperplane Domination Decision Criterion for Spheres)**.** Let $a, b \in \mathbb{R}^d$ be points (spheres with radius 0) and $R \in \mathbb{R}^d$ be a sphere. The *Hyperplane* domination decision criterion for spheres is defined as

$$\mathrm{DDC}_{HP}(a, b, R) \Leftrightarrow R \subset H_{ab}(a).$$

The proofs for correctness and completeness of DDC_{MM} for spheres are analogous to the ones for rectangles (cf Proof 3.2.1 and 3.2.1 respectively).

Regarding the efficiency of $\mathrm{DDC}_{HP}(a, b, R)$ for spheres again the situation is similar to the rectangular case. In order to test if a sphere R is completely contained in the halfspace $H_{ab}(a)$ the distance of the sphere center $R.c$ to the plane H_{ab} has to be computed and compared with the radius $R.r$ of R. In particular the right-hand side of the following statement has to be computed

$$\mathrm{DDC}_{HP}(a, b, R) \Leftrightarrow dist(R.c, H_{ab}) < R.r$$

where $dist()$ is the Euclidean distance function between point and plane, which can be computed in $O(d)$, where d is the dimension of $R.c$ and H_{ab}.

6.1.2 The Criterion

In this section we will show how to build a correct and complete domination decision criterion (in order to detect $Dom(A, B, R)$) for spheres which is also efficiently computable. For this purpose we will proceed as follows: First we will focus on the 2-dimensional case where A and B are points and R is a circle. For this case we will show that for deciding domination it is sufficient to only consider the surface points of the circle R (and not all points contained in R). Second we will extend the criterion to higher dimensions ($d > 2$). Third we explain how to efficiently compute the criterion and last but not least we will show how to handle extended spherical approximations of A and B.

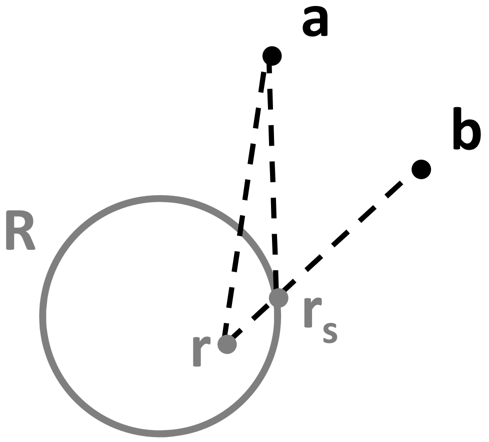


Figure 6.2: Considering the surface points of R is sufficient

Preliminaries

Let us first show that for deciding domination it is sufficient to consider only the points on the surface of R for the case where A and B are points. Let us note that this case is already handled by the *Hyperplane* domination decision criterion, however we will stepwise extend this technique for the basic case to more general cases.

Lemma 6.1. Let a and b be points, R be a circle in $\mathbb{R}^2$,$R_s = \{r_s | dist(R.c, r_s) = R.r\}$ be the points on the surface of R and $dist()$ be the Euclidean distance function, then

$$Dom(a, b, R) \Leftrightarrow \forall r \in R : dist(a, r) < dist(b, r) \Leftrightarrow \forall r_s \in R_s : dist(a, r_s) < dist(b, r_s) \tag{6.1}$$

Proof. Assume there exists a point $r \in R$ such that $dist(a, r) > dist(b, r)$, then let $r_s \in R_s \wedge r_s \in \overline{R.ca}$ be the point on the intersection of R_s and the line connecting $R.c$ and a (cf Figure 6.2). Then if

$$dist(a, r) > dist(b, r)$$

due to the triangle inequality it holds that

$$dist(a, r_s) + dist(r_s, r) > dist(b, r_s) + dist(r_s, r)$$

which can be simplified to

$$dist(a, r_s) > dist(b, r_s)$$

□

The proof shows that if there exists a point $r \in R$ for which $dist(a, r) < dist(b, r)$ then there also exists a point $r_s \in R_s$ for which $dist(a, r_s) < dist(b, r_s)$ holds. Consequently it is sufficient to only consider the points $r_s \in R_s$ for deciding if $Dom(a, b, R)$ holds.

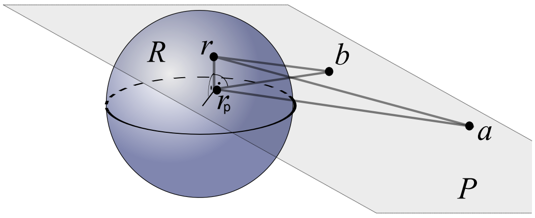


Figure 6.3: Considering only points in P and R is sufficient

Extension to the Multi-Dimensional Case

Next we want to extend Lemma 6.1 to the multi-dimensional case.

Lemma 6.2. Let a and b points, R be a sphere in $\mathbb{R}^d$, P be the two dimensional plane defined by the three points a, b and $R.c$. Furthermore let $r \in R$ be a point and $dist()$ be the Euclidean distance function, then

$$\exists r \in R : dist(a, r) > dist(b, r) \Leftrightarrow \exists r_p \in P : dist(a, r) > dist(b, r)$$

Proof. Consider Figure 6.3 as illustration. Let $r \in R$ be a point for which $dist(a, r) > dist(b, r)$. Furthermore let the point r_p be the projection of r onto the plane P [1], then the following holds:

$$dist(a, r) > dist(b, r)$$

using Pythagoras' theorem (which is applicable here due to the right triangle resulting from the projection) we can rewrite this as

$$dist(a, r_p)^2 + dist(r, r_p)^2 > dist(b, r_p)^2 + dist(r, r_p)^2$$

thus it holds that

$$dist(a, r_p) > dist(b, r_p)$$

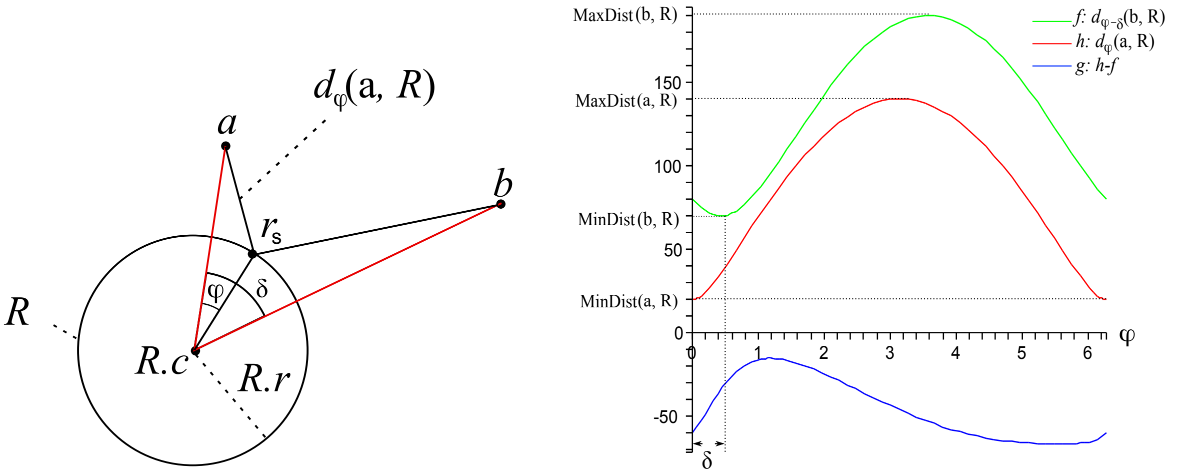
□

The above Lemma 6.2 states that if there exists a point $r \in R$ which is closer to point a than to point b, then there does also exist a point $r_p \in P$ in the plane defined by a, b and $R.c$ for which the same holds. Thus this lemma lets us reduce the multi-dimensional case of finding such a point $r \in R$ to the two dimensional case of the projection on plane P.

Combining the implication of Lemma 6.1 and Lemma 6.2 we can now state that if we cannot find a point r_s on surface of the circle defined by the intersection of the plane P and the sphere R, for which $dist(a, r_s) > dist(b, r_s)$ then no point $r \in R$ can exist for which $dist(a, r) > dist(b, r)$ holds. For ease of reading and w.l.o.g. we will in the following explain all techniques for the 2-dimensional case, which only considers the projection of A, B and R onto the plane P defined by $A.c, B.c$ and $R.c$ respectively.

---

[1] Let us note that this projection always results in a point $r_p \in R$i, since circle resulting through the cut through the center of a sphere is always the largest possible circle



(a) Illustration of the surface distance

(b) Distance functions $d_\varphi(a, R)$ and $d_\varphi(b, R)$ of two points a and b and sphere R with $\delta = \measuredangle aR.cb$

Figure 6.4: Trigonometric domination computation

Efficient Computation

Though we could constraint the number of points to be considered for deciding $Dom(a, b, R)$ the remaining problem is still how to efficiently decide the right-hand side of Equality 6.1. Aiming at efficient computation we are going to define the *surface distance*.

Definition 6.4 (Surface Distance $d_\varphi(p, R)$)**.** Let R be a sphere around the center point $R.c$ with radius $R.r$. Let $r_s \in R_s$ be a point on the surface of R, i.e. $dist(R.c, r_s) = R.r$ and let $p \in \mathbb{R}^d$ be a point. The *surface distance* $dist(p, r_s)$ follows the function

$$d_\varphi(p, R) = \sqrt{dist(R.c, p)^2 + R.r^2 - 2\, dist(R.c, p)\, R.r\, \cos(\varphi)}\ ,$$

where $\varphi = \measuredangle pR.cr_s$ is the angle between $\overrightarrow{R.cp}$ and $\overrightarrow{R.cr_s}$. If p is equal to the center $R.c$, φ is not defined, thus we set $d_\emptyset(p, R) = r_{\mathcal{R}}$ for $p = R.c$.

With the *surface distance* it is possible to formulate the following lemma.

Lemma 6.3. Let a and b points and R be a sphere in $\mathbb{R}^2$ then

$$Dom(a, b, R) \Leftrightarrow \forall \varphi \in [0, 2\pi] : d_\varphi(a, R) < d_{\varphi-\delta}(b, R)$$

where $\delta = \measuredangle aR.cb$. This implies that $d_\varphi(a, R)$ and $d_\varphi(b, R)$ are the distances between a and a point $r_s \in R_s$ and b and the exact same point r_s, respectively.

Proof. The above lemma basically tests for each point $r_s \in R_s$ on the surface of R if $dist(a, r_s) < dist(b, r_s)$. If this condition holds then $Dom(a, b, R)$ is also true according to Lemma 6.1. □

An illustration of the surface distance and how it is utilized for trigonometric domination is given in Figure 6.4. Evaluating the inequality on the right-hand side of Lemma 6.3 efficiently involves several steps. First we rewrite the inequality to

$$Dom(a,b,R) \Leftrightarrow \forall\varphi \in [0,2\pi] : d_\varphi(a,R) - d_{\varphi-\delta}(b,R) < 0$$

Then we can define a new function $g_\varphi(a,b,R)$ which is given the difference function:

$$g_\varphi(a,b,R) = d_\varphi(a,R) - d_{\varphi-\delta}(b,R) \tag{6.2}$$

A root of $g_\varphi(a,b,R)$ now indicates that $d_\varphi(a,R)$ is not smaller than $d_{\varphi-\delta}(b,R)$ meaning that $Dom(a,b,R)$ does not hold (cf Lemma 6.3). Finding the roots of $g_\varphi(a,b,R)$ is computational quite expensive, however, we only need to know if there exists a root and not the exact value. Thus we define a second function which has the same square roots:

$$k_\varphi(a,b,R) = (d_\varphi(a,R))^2 - (d_{\varphi-\delta}(b,R))^2$$

Obviously the function k has it's roots at the same values of φ since the surface distances are always positive. For detecting if $k_\varphi(a,b,R)$ has roots we calculate the extrema by examining the roots of $k'_\varphi(a,b,R)$ as defined in Equation 6.3.

$$\begin{aligned} k'_\varphi(a,b,R) &= [(d_\varphi(a,R))^2]' - [(d_{\varphi-\delta}(b,R))^2]' \\ &= 2\cdot dist(R.c,a)\cdot R.r\cdot sin(\varphi) - 2\cdot dist(R.c,b)\cdot R.r\cdot sin(\varphi-\delta) \end{aligned} \tag{6.3}$$

$$k'_\varphi(a,b,R) = 0 \Rightarrow \varphi_{1,2} = arctan(\frac{dist(R.c,a)\cdot sin(i\pi+\delta)}{dist(R.c,b)\cdot cos(i\pi+\delta) - dist(R.c,a)})$$

with $i \in \{0,2\}$ yielding a maximum of 2 extrema points. Using the signum function, we can now differentiate between the following cases:

1. $\text{sgn}(g_{\varphi_1}(a,b,R)) = \text{sgn}(g_{\varphi_2}(a,b,R)) < 0$:
   $\Rightarrow \forall\varphi$: $g_\varphi(a,b,R) < 0$ and thus $\forall\varphi$: $d_\varphi(a,b,R) < d_{\varphi-\delta}(a,b,R) \Rightarrow Dom(a,b,R)$.
2. Otherwise:
   $\Rightarrow \exists\varphi\in[0;2\pi[$: $d_\varphi(a,b,R) > d_{\varphi-\delta}(a,b,R) \Rightarrow Dom(a,b,R)$ does not hold.

Extended Approximations

Having the theoretical base of Lemmas 6.1 - 6.3 it is now easy to extend the proposed criterion to the case where A and B are spheres as well.

Lemma 6.4. Let A, B and R be spheres in $\mathbb{R}^d$ and let R' be the projection of R on the plane P defined by $A.c, B.c$ and $R.c$, respectively, then

$$Dom(A,B,R) \Leftrightarrow \forall\varphi \in [0,2\pi] : (d_\varphi(A.c,R')) + A.r < (d_{\varphi-\delta}(B.c,R')) - A.r$$

Proof. In order to show that $Dom(A, B, R)$ holds we have to show that

$$\forall a \in A, b \in B, r \in R : dist(a, r) < dist(b, r)$$

holds. Which can be reformulated to

$$\forall r \in R : MaxDist(A, r) < MinDist(B, r)$$

The MaxDist (MinDist) between a point and a sphere can be computed by just adding (subtracting) the radius of the sphere to the distance between the point and the center of the sphere. This lets us straightforwardly reformulate Lemma 6.3 for the case of extended objects A and B. Additionally 6.1 and Lemma 6.2 let us restrict the search space of points r to points $r_s \in R'$. □

Last but not least we have to show that 6.4 can be computed efficiently. Consider the following function which is the adapted version (without squares and including radii) of the function defined in Equation 6.2 for extended objects:

Lemma 6.5. Let A, B and R be spheres in $\mathbb{R}^d$ and let R' be the projection of R on the plane P defined by $A.c, B.c$ and $R.c$. Furthermore let $g_\varphi(A, B, R)$ be the following difference function

$$g_\varphi(A, B, R) = d_\varphi(A.c, R') + A.r - d_{\varphi-\delta}(B.c, R') - A.r$$

then $g_\varphi(A, B, R)$ has it's roots for the same values of φ than $g_\varphi(A.c, B.c, R)$ (which was defined in Equation 6.2)

Proof. Consider Figure 6.4(b). When A and B are spheres with radius > 0 then the surface distance function between $A.c$ and R has to be increased by $A.r$ and the surface distance function $B.c$ and R has to be decreased by $B.r$ at each $\varphi \in [0 : 2\pi[$. This only results in a shift of the difference function $k_\varphi(A.c, B.c, R)$ parallel to the y-axis and has thus no effect on the φ values of the roots of the function. □

Definition 6.5 (Trigonometric domination decision criterion)**.** Let A, B and R be spheres in $\mathbb{R}^d$ and let R' be the projection of R on the plane P defined by $A.c, B.c$ and $R.c$. Then

$$\mathrm{DDC}_{TR}(A, B, R) \Leftrightarrow \mathrm{sgn}(g_{\varphi_1}(A.c, B.c, R') + A.r) = \mathrm{sgn}(g_{\varphi_2}(A.c, B.c, R') - A.r) < 0$$

where φ_1 and φ_2 are the roots of the function $g_\varphi(A.c, B.c, R')$ which are computed using the techniques proposed in Section 6.1.2.

The correctness of the *Trigonometric* domination decision criterion follows directly from the equivalence relations in Lemmas 6.1 - 6.5. Also DDC_{TR} is efficiently computable since it only involves two distance computations and a constant number of trigonometric and more simple operations independent of the dimensionality of the space.

6.2 Application: All-Nearest-Neighbour Queries

In the preliminaries of this work we emphasised that similarity search in databases is an important problem in content-based multimedia retrieval. Additionally, similarity queries are a useful database primitive for speeding up multiple data mining algorithms on large databases. One of the most important types of similarity queries in this setting are k-nearest neighbor (kNN) queries, retrieving the k objects in the database which have the smallest distance to a given query vector q. The majority of the approaches developed so far focus on the efficient processing of a single query at a time. However, in many applications like data mining and similarity search, it is previously known that it is necessary to process a large number of kNN queries to generate a result.

More precisely, an All k nearest neighbour (AkNN) query retrieves the k-nearest neighbors in the inner set or database $\boldsymbol{S}$ for each object in the outer or query set $\boldsymbol{R}$. Let us note that the same type of query is also known as kNN join [39]. Multiple computational problems use AkNN queries: In multimedia retrieval, many recent approaches model the image content as a set of local descriptors [115] or point clouds [139]. An AkNN query efficiently retrieves the best matches for a set of local descriptors in a given image database. Another area of application is data mining: [39] surveys multiple data mining algorithms that can be accelerated by efficient methods for AkNN computation like parallel kNN classification and k-means clustering. Furthermore, it is possible to employ AkNN processing for deriving outlier factors like [38].

In this chapter, we propose a new approach for processing AkNN queries. As our method uses spherical page regions it employs an SS-Tree [166]. To define the pruning area around the approximations, we utilize the *Trigonometric* domination decision criterion which accounts for the fact that the relevant search space is not necessarily symmetric around the query approximation. One of its advantages is its capability to calculate pruning areas which are based on query approximations and to considerably decrease the remaining search space employed by other approaches like [139, 47].

6.2.1 Problem Formalization

In this section, we will formalize our task of AkNN queries and describe the index structure used for our approach.

All-k-Nearest-Neighbors

As mentioned before, AkNN queries aim at retrieving the k-nearest neighbors for each element of a given query or outer set $\boldsymbol{R}$ in the inner data set $\boldsymbol{S}$.

Definition 6.6 (AkNN Query)**.** Let $\boldsymbol{R}, \boldsymbol{S} \subset \mathbb{R}^n$ be two data sets and $dist : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ a distance metric on $\mathbb{R}^d$. An all-k-nearest-neighbor (AkNN) query retrieves the result set $\mathrm{ANN}_k(\boldsymbol{R}, \boldsymbol{S}) \subseteq \boldsymbol{R} \times \boldsymbol{S}$ for any $k \in \mathbb{N}$ for which the following condition holds:

$$\forall\ R \in \boldsymbol{R},\ S \in \boldsymbol{S} \colon (R, S) \in \mathrm{ANN}_k(\boldsymbol{R}, \boldsymbol{S}) \Rightarrow S \in \mathrm{kNN}(R, \boldsymbol{S})$$

Our algorithm for efficiently processing AkNN queries is built on trigonometric functions, which need to be valid in the underlying featurespace. In this chapter we thus rely on the Euclidean metric, the most popular distance metric for this kind of applications. Furthermore, we assume that the inner set $\boldsymbol{S}$ is organized in an index structure which is based on spherical page regions. A spherical page region is specified by a centroid $C \in \mathbb{R}^n$ and a radius $r \in \mathbb{R}^+ \setminus \{0\}$ and thus, it describes a hypersphere around C, containing all points P_i with distance $d(C, P_i) \leq r$. Although approximating page regions using minimum bounding rectangles is more common for spatial index structures, there have been several successful index structures employing spherical page regions [166, 56].

The SS-Tree

As mentioned above, our method employs the SS-Tree [166], an efficient index structure for similarity search based on spherical page regions. In the following, we will shortly review the characteristics of the SS-Tree.

Definition 6.7 (SS-Tree)**.** An SS-Tree in the vector space $\mathbb{R}^d$ is a balanced search tree having the following properties:

- All nodes besides the root store between m and M entries. The root is allowed to store between 1 and M entries.
- The entries of a leaf node are feature vectors in $\mathbb{R}^d$. The entries of an inner node are nodes, i.e. roots of subtrees.
- Each entry is bounded by a spherical page region containing all son entries.
- For a leaf node $\mathcal{L}$, the center $C_{\mathcal{L}}$ of the containing hypersphere $B_{\mathcal{L}}$ is the centroid of all contained feature vectors. The radius of $B_{\mathcal{L}}$, $r_{\mathcal{L}}$, is the maximum distance between any entry vector $L \in \mathcal{L}$ and $C_{\mathcal{L}}$.
- For an inner node $\mathcal{P}$, the center $C_{\mathcal{P}}$ of the containing hypersphere $B_{\mathcal{P}}$, is the centroid of the centroids $C_{\mathcal{Q}}$ of the contained son pages $\mathcal{Q} \in \mathcal{P}$. The radius $r_{\mathcal{P}}$ is chosen as $\max_{\mathcal{Q} \in \mathcal{P}} (d(C_{\mathcal{P}}, C_{\mathcal{Q}}) + r_{\mathcal{Q}})$.

In general, the structure of the SS-Tree is quite similar to the structure of the R-Tree [77]. However, the page approximations are spherical instead of rectangular. Additionally, the split heuristics for creating the tree are different. Instead of minimizing the overlap between two pages, the SS-Tree tries to minimize the variance within the entries of its pages. Thus, when splitting a node, the split heuristics determines the dimension having the largest variance. In this dimension, the partition is selected which minimizes the variance of the resulting pages and for which both new pages contain at least m entries.

n	The dimension of the feature space
$\boldsymbol{R}$	The outer set ($\subseteq \mathbb{R}^n$)
$\boldsymbol{S}$	The inner set ($\subseteq \mathbb{R}^n$)
$\mathcal{R} \subseteq \boldsymbol{R}$, $\mathcal{S} \subseteq \boldsymbol{S}$	Subsets of the outer set or the inner set
$R \in \boldsymbol{R}$, $S \in \boldsymbol{S}$	Points from the outer set or the inner set ($\in \mathbb{R}^n$)
$B_{\mathcal{R}}$, $B_{\mathcal{S}}$	Spheres $\equiv$ blocks around the sets $\mathcal{R}$, or $\mathcal{S}$, respectively ($\in \mathbb{R}^n$)
$\mathcal{P}_{\text{cand}}$	Candidate set of points of the inner set ($\subseteq \boldsymbol{S}$)
$r_{\mathcal{R}}$, $r_{\mathcal{S}}$	Radius of sphere $B_{\mathcal{R}}$ or $B_{\mathcal{S}}$ around set $\mathcal{R}$ or $\mathcal{S}$
$C_{\mathcal{R}}$, $C_{\mathcal{S}}$	Center point of sphere $B_{\mathcal{R}}$ or $B_{\mathcal{S}}$ around set $\mathcal{R}$ or $\mathcal{S}$

Table 6.1: Table of notations

Principles of AkNN Algorithms

Given two data sets $\boldsymbol{R}$ and $\boldsymbol{S}$, where $\boldsymbol{S}$ is organized in an index structure. We want to find the kNNs of all elements $R_i \in \boldsymbol{R}$. The most trivial approach would be to retrieve the kNN for each element R_i separately by using the index structure organizing $\boldsymbol{S}$. Obviously this is not very efficient, e.g. for two query points (in $\boldsymbol{R}$) with a close proximity the same pages of the index have to be read twice. An efficient algorithm needs to group the points in $\boldsymbol{R}$ so that spatially close points fall into the same group. Afterwards these groups, containing distinct subsets of $\boldsymbol{R}$, are used as query sets. For the ensuing step, it is important to decide which pages of $\boldsymbol{S}$ need to be examined for finding the kNN of each element contained in a subset $\mathcal{R} \subseteq \boldsymbol{R}$. Therefore, the search space, i.e. the space which could contain kNNs of an element $R \in \mathcal{R}$, needs to be defined. To successively minimize the search space, most algorithms start with the search space containing $\boldsymbol{S}$. Defining a pruning area has the opposite intention: define the space which does not have to be further examined. This means, that all pages of $\boldsymbol{S}$ lying completely in the pruning area can be discarded as candidates for the subset $\mathcal{R}$. Keep in mind that pages not completely covered by the pruning area can potentially contain points lying in the search space and therefore need to be resolved. Since all elements in the search space have to be considered in the further processing of $\mathcal{R}$, the goal is to minimize the search space as strongly and quickly as possible. Minimizing the search space, respectively maximizing the pruning area, is the task of a pruning criterion. The optimal search space would only contain the kNNs of all $R \in \mathcal{R}$. Therefore, a pruning criterion should try to estimate this optimum as well as possible with low effort. For ease of readability we provide a table of notations for this chapter (cf Table 6.1).

6.2.2 Using Domination Decision Criteria for AkNN

In this Section we will show how to use the domination relation ($Dom(A, B, R)$) as a pruning criterion for the problem of AkNN queries. In the following we first start with the basic case of $k = 1$ and will later discuss the case of $k > 1$.

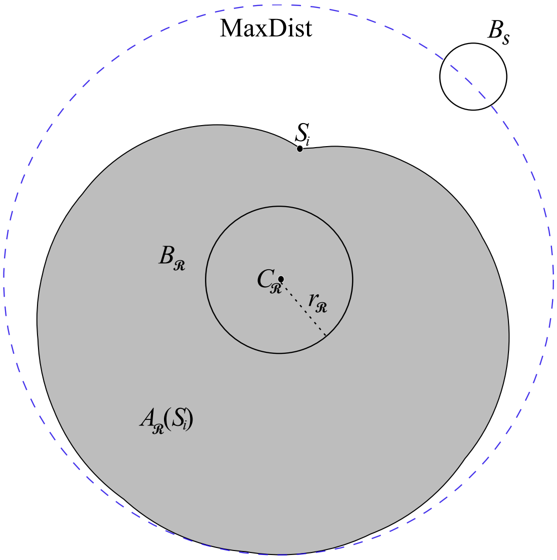


Figure 6.5: Domination for AkNN query processing

Lemma 6.6. Let $\mathcal{R} \subseteq \boldsymbol{R}$ be as set of points from the outer set and $S_i \in \boldsymbol{S}$ be a point from the inner set, then the set of points $\mathcal{S} \subseteq \boldsymbol{S}$ can not be part of the result of an A1NN query if $Dom(S_i, B_{\mathcal{S}}, B_{\mathcal{R}})$ holds and can thus be pruned.

Proof. The following holds by definition of the domination relation: $Dom(S_i, B_{\mathcal{S}}, B_{\mathcal{R}}) \Rightarrow \forall S_j \in B_{\mathcal{S}}, R_k \in B_{\mathcal{R}} : dist(S_i, R_k) < dist(S_j, R_k)$. Since the point S_i is closer to each point $R_k \in B_{\mathcal{R}}$ than any point $S_j \in B_{\mathcal{S}}$ no point R_k can possibly have a point S_j as nearest neighbour. ☐

Using Lemma 6.6 it is possible to reduce the search space (the space which may possibly contain results of the query). This remaining search space is often also called *active region*.

Definition 6.8 (Active Region for A1NN)**.** Let $\mathcal{R} \subseteq \boldsymbol{R}$ be as set of points from the outer set and $PC \subseteq \boldsymbol{S}$ be a set of points from the inner set then the active region is defined by

$$A_{\mathcal{R}}(PC) = \{P \in \mathbb{R}^d | \forall S_i \in PC : \neg DDC(S_i, P, B_{\mathcal{R}})\}$$

The active region thus represents the part of the whole data space which could potentially contain a nearest neighbour for one of the points contained in $\mathcal{R}$. Every object which does not intersect with this *active region* can thus be pruned for a $A1NN$ query. Obviously the shape and extension of the *active region* varies depending on the used domination decision criterion. Figure 6.5 illustrates the *active region* $A_{\mathcal{R}}(\{S_i\})$ for DDC$_{TR}$ (grey shaded area) and for DDC$_{MM}$ (area in the dotted circle) for a 2-dimensional example. In this example the set of points contained in $B_{\mathcal{S}}$ can be pruned according to DDC$_{TR}$ but cannot be pruned when using DDC$_{MM}$.

Extending the idea of using the domination decision criteria from A1NN to AkNN is very simple: A point or spherical page (containing several points) can be pruned if it is dominated by at least k points from the inner set $\boldsymbol{S}$ according to a query set $\mathcal{R} \subseteq \boldsymbol{R}$.

6.2.3 Related Work

There already exist several approaches for processing AkNN queries. The initial and most simple approach is the computation of a separate kNN query on the inner set $\boldsymbol{S}$ for each point in the outer set $\boldsymbol{R}$. This method is often referred to as nested loop join and can be accelerated by various search methods for kNN query processing. A comprehensive survey can be found in [138]. Since this basic approach obviously is not optimal w.r.t. the number of page accesses and distance computations, several dedicated approaches for processing AkNN queries have been proposed. We first of all have to distinguish index-based from scan-based solutions. In a scan-based solution, we assume that both data sets are not organized in any form of index and thus, we have to scan the complete data set for join processing. Examples for processing kNN joins without the use of index structures are GORDER [169] and HANN [176].

In this chapter, we assume that at least $\boldsymbol{S}$ is already organized in an index structure. It is shown in the experiments in [47, 176] that the use of an index structure can considerably speed up kNN join processing. One of the first publications discussing kNN join algorithms was [39]. In this paper, Böhm et al. demonstrate that kNN joins are useful database primitives that can be employed to speed up various data mining algorithms like k-Means clustering. The proposed algorithm is based on a new data structure called multipage index (MuX) which introduces larger pages for fast I/O accesses which are further organized into memory pages aiming at minimizing distance calculations. The proposed join algorithm first retrieves the current page of $\boldsymbol{R}$ and then queries $\boldsymbol{S}$ with the set of all query points. For each query point, the algorithm maintains its own active page list. The current pruning distance is considered as the maximum element of any query of these queues, and pages are refined w.r.t. the minimal distance to any query element. A drawback of this approach is that it has an overhead of distance computations because each object in $\boldsymbol{S}$ has to be compared to all elements in the query page. In [176] the authors discuss two methods for AkNN queries that assume that $\boldsymbol{S}$ is organized in a spatial index structure like the $\boldsymbol{R}$-Tree [77]. The first index-based approach discussed in [176] aims at improving the use of a disk cache when posing a separate NN query for each query point. The second proposed method, called *batched nearest neighbor* search (BNN), groups query points based on a Hilbert curve and poses one query for each group of points. The method considers the distance to the k-nearest neighbor $\delta_{k,x}$ for each element x in the current query set. The maximum of these distances is now used to extend the minimum bounding rectangle around the query set and thus, describes the current pruning area. In [173], a kNN join algorithm based on the similarity search method iDistance [87] is proposed. The paper describes a basic algorithm called iJoin extending iDistance to the kNN join problem. Furthermore, two extensions are proposed that employ MBR-approximations to reduce distance computations and a dimensionality reduction approach to improve the performance on higher dimensionalities. Recently, two approaches have been published which reduce distance calculations by not considering the particular elements in the current set of query points. Instead, the pruning area is established on a minimum bounding rectangle around the query set. [139] presents an approach for applications on large point clouds in image processing. In this method, the set of query points is approximated by a minimum bounding rectangle (MBR) which is posed as a query to an $\boldsymbol{R}$-Tree organizing $\boldsymbol{S}$. The authors propose to employ the maximum distance that two points in two compared MBRs might have to determine the current pruning range. This criterion is named MaxMaxDist. The authors prove that their algorithm is optimal w.r.t. the number of pages that intersect a query area spanned by the MaxMaxDist. The experiments are focused on the problem of point clouds and thus, are directed at rather low dimensional settings. In [47], the authors propose NXNDIST which is based on the observation that at each side of an MBR, there must be exactly one contained data point which realizes the minimum distance. Thus, NXNDIST decreases the MaxMaxDist by allowing the use of the minimal MaxDist for exactly one dimension. The result is a closer bound for MBRs that is guaranteed to contain at least one nearest neighbor to any query point in the query MBR in the intersecting pages of $\boldsymbol{S}$. In their solution (called MBA in this context), the authors additionally propose to employ MBR-quadtrees as an index structure for $\boldsymbol{S}$ instead of R-Trees.

In contrast to all discussed approaches, our method uses spherical page approximations instead of MBRs. We employ the SS-Tree [166] for indexing $\boldsymbol{S}$. Our main pruning method which is based on the *Trigonometric* domination decision criterion describes the remaining search space based on the query approximations only. However, opposed to previous approaches, the pruning area is not built by symmetrically extending the query approximation in each direction. Instead, we propose the use of an asymmetric pruning area to further limit the search space. Finally, we show how our new approach can be effectively combined with existing pruning methods to achieve a general improvement on data sets of varying characteristics.

6.3 Performing AkNN Queries

Algorithm 5 gives an overview of our query algorithm. In the following, we explain the employed data structures and describe the complete algorithm. Finally, we propose an extension for employing multiple pruning techniques.

The general idea of our algorithm is to preprocess the set $\boldsymbol{R}$ and build several compact groups of points each approximated with a sphere. Each of these groups is then processed separately. For a group of points $\mathcal{R} \subseteq \boldsymbol{R}$ we traverse the index organizing $\boldsymbol{S}$ in a best first manner. Additionally we keep a list of points *PCLIST* containing pruning candidates (e.g. points from the set $\boldsymbol{S}$ which can be used to prune other elements from $\boldsymbol{S}$). For each entry e which is considered during the traversal of the index we check if we find k pruning candidates which dominate the element according to $B_{\mathcal{R}}$. Whenever we find a point during the traversal of the index it may be added to the pruning candidates. However adding all points found in this way to the pruning candidates yields a large list of points which may result in computational overhead. Thus the question is which points to keep during processing. Generally, points close to the query region result in a smaller active region, so that these points are preferable. The active region is further reduced by comparing a point S_j to all pruning candidates $S_i \in \mathcal{S}_{cand}$ using the domination relation, since the resulting active region is equivalent to an intersection of the active regions of all S_i. Hence, more candidate points result in fewer page accesses at the price of an increasing number of distance calculations. The largest benefit is achieved if the pruning candidates are equally distributed around C_S because the resulting active region is minimized in this case. Another important issue is that at least k pruning candidates have to exist before defining a search space smaller than the whole space $\mathbb{R}^d$. Resulting from these findings, we always keep those points with minimal MinDist to the query as pruning candidates. Additionally we define the parameter ϵ, which controls the maximum number of pruning candidates that should be considered for pruning elements from $\boldsymbol{S}$, by setting this number to $k \cdot \epsilon$.

Algorithm 5 AkNN($LIST_{\text{queryRegions}}$, $SSTREE_{\text{innerSet}}$, k, RES_HT)

```
for all $\mathcal{R} \in LIST_{\text{queryRegions}}$ do
  PCLIST := new PCLIST($\epsilon \cdot k$)
  PQ := new PQ()
  PQ.add($SSTREE_{\text{innerSet}}$.root, -1)
  while PQ $\neq \emptyset$ do
    if PCLIST.size = $\epsilon \cdot k$ and PQ.smallestDistance > PCLIST.MaxDist$_k$ then
      break
    end if
    node := PQ.removeFirst()
    if not canBePruned(node, $\mathcal{R}$, PCLIST) then
      for all child $\in$ node do
        if child is an inner node then
          PQ.add(child, MinDist(child, $\mathcal{R}$))
        else if not canBePruned(node, $\mathcal{R}$, PCLIST) then
          PCLIST.add(child)
          for all queryPoint $\in \mathcal{R}$ do
            RES_HT.add(queryPoint, child)
          end for
        end if
      end for
    end if
  end while
end for
```

6.3.1 Data Structures

Algorithm 5 requires the following data structures:

PQ: A priority queue handling all yet unconsidered pages for a query set $\mathcal{R} \subseteq \boldsymbol{R}$. PQ is organized as a heap with the element with the smallest MinDist to the query region on top.

RES_HT: A hash table storing one priority queue for each query point. Each queue has a capacity of k and maintains the k-nearest neighbors for its query point. The call RES_HT.add($\mathcal{R}$, S_i) adds S_i to the priority queue of $\mathcal{R}$.

PCLIST: A list containing the pruning candidates, ordered by their MinDist to the query region. The maximum size of the list is defined by $\epsilon \cdot k$ with k being the amount of nearest neighbors that should be computed and $\epsilon \geq 1$ being an adjustable parameter. The larger ϵ is chosen, the more pages can be pruned (as PCLIST grows and provides more pruning power) which saves time consuming I/O-accesses at the cost of distance calculations.

6.3.2 The A*k*NN Algorithm

Our algorithm starts with grouping the query set $\boldsymbol{R}$ into compact spherical approximations. We use BIRCH [178] to produce a list of compact regions in linear time and organize them in a list. This way, the resulting query regions can be described by compact hyper spheres having a given maximum radius. Since the pruning area also decreases with the radius of a query region, we use the algorithm proposed in [17] to calculate the approximate smallest enclosing ball (SEB) for the data content of each region. Computing SEBs costs extra CPU time, but it pays off fast with growing $|\boldsymbol{S}|$, as the resulting SEBs' radii are about 10% - 20% smaller than the radii of the original spheres.

The main part of the algorithm (cf Algorithm 5) now receives a list of query regions from $\boldsymbol{R}$ and the number of nearest neighbors k as input and proceeds as follows: The query regions are processed successively and independently from each other. First, a new query region is taken from the list. Afterwards, a list for storing pruning candidates (PCLIST) and a priority queue (PQ) are initialized and the root of the SS-Tree storing $\boldsymbol{S}$ is put into the page list PQ. The pages in PQ are ordered by the MinDist to the currently processed query region ($B_{\mathcal{R}}$). The SS-Tree is then traversed in a best-first manner, as proposed by Hjaltason and Samet [82]. This is done by always removing the first node from the priority queue PQ processing it and putting all child nodes (if available) back to PQ, as long as the queue is not empty or the stopping criterion is triggered: The search for other kNN-candidates can be stopped if the MinDist of the current node to $\mathcal{R}$ is larger than the MaxDist of the k-th pruning candidate. In that case, the node and all its successors in the PQ can be pruned.

Algorithm 6 canBePruned(node, $\mathcal{R}$, PCLIST, k)

```
count := 0
for all $S \in PCLIST$ do
    if $DDC(S, B_{node}, B_{\mathcal{R}})$ then
        count := count+1
    end if
    if count $\geq k$ then
        return true
    end if
end for
return false
```

If the algorithm cannot be terminated in this way, it tests whether the current node

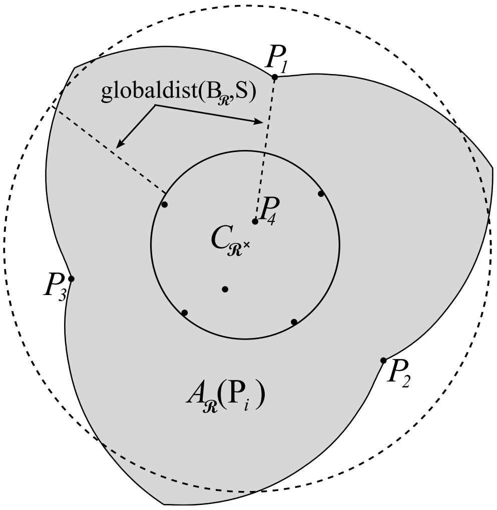


Figure 6.6: The gray shaded area shows the active region used by trigonometric pruning (see Definition 6.8) which results from the intersection of the three pruning areas of P_1, P_2, P_3. The dashed line $[P1, P4]$ indicates the GlobalDist$(B_{\mathcal{R}}, S)$ which defines the pruning area (dashed outer circle) used by BNN in case of a 1NN query.

can be pruned. This is done by the canBePruned() function (see Algorithm 6), which prunes the node or point w.r.t. the pruning techniques introduced in Sect. 6.2.2. If the node cannot be pruned and its children are nodes of the SS-Tree, they are all added to the page list PQ. If the children are data points, another test on whether or not they can be pruned ensues. If the test is negative, they are added to the PCLIST and to all priority queues (via RES_HT) of query points contained in $\mathcal{R}$. Let us note that it is possible that the points will be eliminated from both data structures at a later point of time.

6.3.3 Combining Trigonometric Pruning with Other Pruning Criteria

In this section, we will discuss the use of different pruning criteria to further increase the performance. Even if employing additional pruning techniques causes additional computational cost, the cost is negligible if the combination yields a significant decrease of the search space and thus saves page accesses. In order to maximize the effect of combining different pruning criteria, it is important that the combined criteria perform always at least as well as one criterion alone. In our case, this can be achieved when trigonometric pruning (TP), which is based on DDC$_{TR}$, is combined with a pruning criterion that also defines a pruning region so that the intersection of both regions can be used as a new pruning region.

In our experiments, we combine TP with BNN [176] which turned out to be one of the best pruning criteria we examined. The fact that the original BNN algorithm is proposed to be applied on rectangular page regions is not a problem, as it can be transferred to spherical page regions, such that BNN can be applied to the SS-Tree as well. The authors of BNN propose to query the inner set $\mathcal{S}$ with compact groups $B_{\mathcal{R}i} \in \mathcal{R}$ whereas each $R_i \in B_{\mathcal{R}i}$ organizes its NNs and according maximum-NNDist$(R_i, \boldsymbol{S})$. The largest value of NNDist$(R_i, \boldsymbol{S})$ of all $R_i \in B_{\mathcal{R}i}$ defines the GlobalDist$(B_{\mathcal{R}i}, \boldsymbol{S})$ of $B_{\mathcal{R}i}$. Pages $\mathcal{S}$ can then be pruned if MinDist$(B_{\mathcal{R}i}, \mathcal{S}) >$ GlobalDist$(B_{\mathcal{R}i}, \boldsymbol{S})$ as $\mathcal{S}$ cannot contain a NN for any $R_i \in B_{\mathcal{R}i}$. The disadvantage of the BNN algorithm compared to TP is that the spatial relation to other NN candidates is ignored. This can possibly lead to the case where GlobalDist(C, S) degenerates and converges to MaxDist in the worst case. An example of the remaining search space when combining both pruning criteria is illustrated in Figure shown in Fig. 6.6.

6.4 Experimental Evaluation

In the following, we outline the evaluation process of the methods suggested in this chapter. We compare them w.r.t. to I/O cost to the state-of-the-art methods BNN [176] and MBA [47].

6.4.1 Experimental Setup

In order to make the approaches competitive for data sets of larger dimensionality, we adapted the algorithm to the X-Tree [26] and the SS-Tree [166] instead of using an R-Tree [77]. We display results for the following algorithmic settings:

MP	AkNN-algorithm (see Algorithm 5), using DDC$_{MM}$ in the canBePruned() function
XBNN	BNN, using **S** indexed in an X-Tree
SSBNN	BNN, using **S** indexed in an SS-Tree
TP	AkNN-algorithm (see Algorithm 5), using DDC$_{TR}$ in the canBePruned() function
TP+SSBNN	AkNN-algorithm combining TP and BNN in an SS-Tree
MBA	MBA algorithm [47]

We have evaluated the algorithms on the following three real-world data sets, which were also used in the evaluation of [47]. The first dataset is the Twin Astrographic Catalog (TAC) Version 2 [175] which is a library of stellar coordinates resulting in a set of 705 099 two-dimensional star representations. The second dataset is the Forest Cover Type (FC), retrieved from the UCI KDD repository [80] which consists of 581 012 data points with 10 real-valued attributes, each representing a 30x30 meter square of a forest region. The third dataset is are Corel color histograms (COREL), which is also available at the UCI KDD repository [80]. It consists of 32-dimensional color histograms for 68 040 images.

For all experiments we used a third of the data sets as outer set, the rest as inner set. The page size was set to 1KB for TAC and 4KB for FC and COREL. All experiments were run on a 64bit Intel$^{\circledR}$ Xeon$^{\text{TM}}$CPU (3GHz) with 16G RAM, under Microsoft Windows 2003 R2, with Service Pack 1.

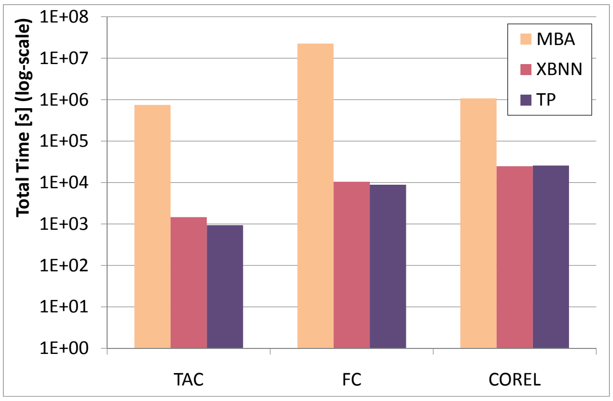


Figure 6.7: Comparison of MBA, XBNN and TP for $k = 10$. The figure displays the sum of I/O and CPU times for one AkNN query.

6.4.2 Results

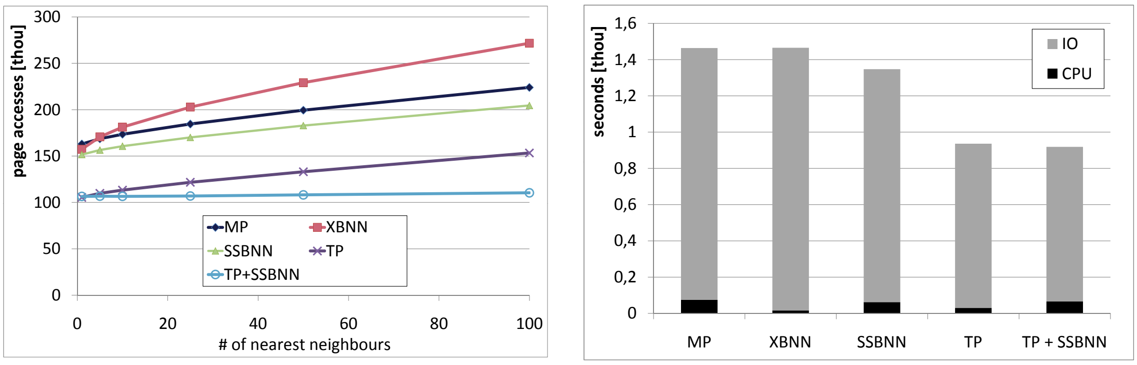
In this section, we present the results of our experimental evaluation for the named algorithms and data sets. Additionally, we will describe the effect of the employed parameters on the query performance.

Comparison to Other Approaches

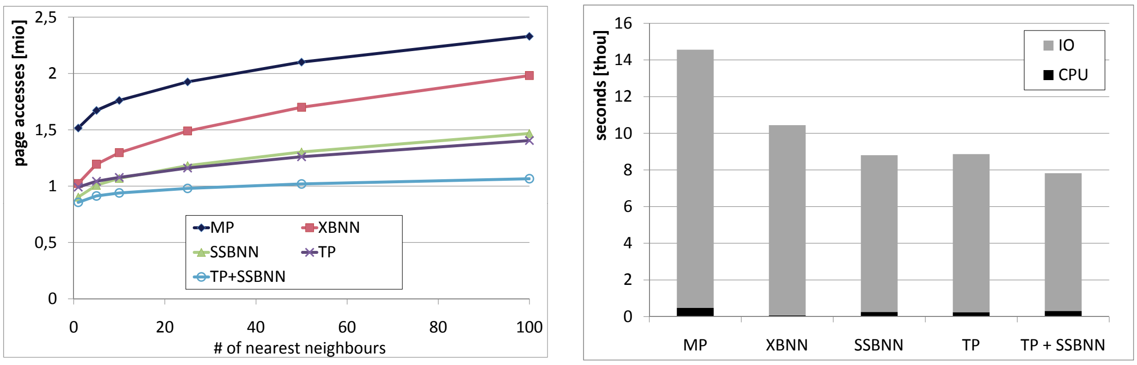
In order to compare the different approaches, we performed all-10-NN queries with the mentioned AkNN-algorithms on the three data sets and measured the CPU-time and the number of page accesses. To combine these two measures, we considered each logical page access with 8ms. Fig. 6.7 illustrates that TP and BNN perform orders of magnitudes better than MBA on all used data sets in matters of the overall runtime. Similar results were obtained with the parametric settings used for the experiments described in the following paragraphs. Therefore, we excluded the results of MBA in the further diagrams because the required scaling would conceal interesting effects.

Dependency on k

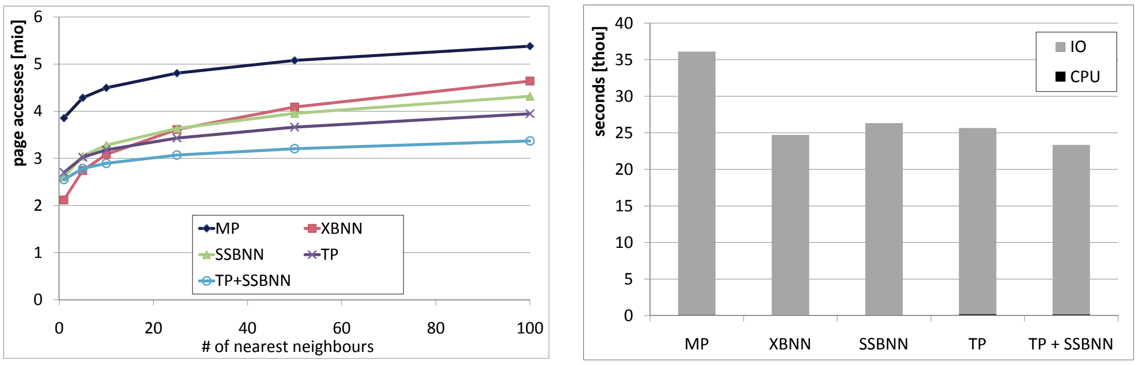
In Fig. 6.8 (left) we compare the named algorithm w.r.t. the number of retrieved neighbors k on all three data sets. Though most methods compare their performance on rather small values of k, larger values of k are often of large practical relevance. In the context of search engines, it is quite common to provide large result sets which are ordered with respect to the similarity to the query. Since the performance mainly relies on the I/O operations, only page accesses are measured. While TP and BNN perform comparable for smaller values of k, TP clearly shows a better performance for larger values of k. For all data sets, our new combined approach (TP + SSBNN) performed significantly better than all compared approaches.



(a) AkNN on TAC (b) A-10-NN on TAC



(c) AkNN on FC (d) A-10-NN on FC



(e) AkNN on COREL (f) A-10-NN on COREL

Figure 6.8: I/O (left) and CPU (right) experiments for AkNN queries on different data sets by all algorithms

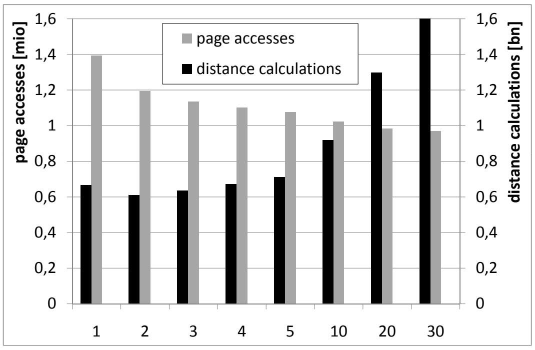


Figure 6.9: ϵ of TP for an A-10-NN query on FC.

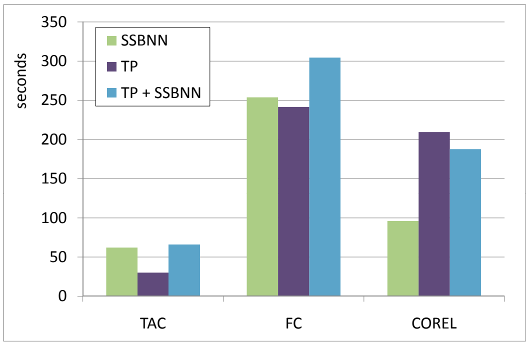


Figure 6.10: CPU-times on different data sets.

Dependency on the Choice of ϵ

The ϵ parameter regulates the number of pruning candidates and thus, it is the most important parameter for tuning the AkNN-algorithm based on TP. Larger ϵ cause a smaller search space by the price of an increased number of distance calculations. In Fig. 6.9, we summarize the effect of ϵ on the performance. As expected, the number of distance calculations grows with increasing ϵ due to the larger number of pruning candidates. Contrarily, more pruning candidates lead to a tighter pruning effect, and thus the number of page accesses is reduced when increasing ϵ. For all experiments in this chapter we set $\epsilon = 5$.

Overall Runtime Performance

The direct comparison in matters of runtime is shown in Fig. 6.8 and yields the following insights:

Comparing MP and TP shows that TP reduces the page accesses by 30% - 40% in contrast to MP. This is a consequence of the reduction of the search space as shown in Fig. 6.5. This effect is very stable over all data sets, suggesting its independence from the amount of dimensions.

The implementations of BNN on different underlying index structures (XBNN and SSBNN) show that the SS-Tree is at least comparable to the X-Tree in the performed experiments. Only for the 32-dimensional COREL data set, the X-Tree leads to a slightly better performance. Considering that the SS-Tree is not as suitable for large dimensions as the X-Tree, these results demonstrate that spherical page regions are well-suited for the AkNN problem.

The effect of TP compared to BNN is dependent on the data set. On the TAC data set, TP clearly outperforms SSBNN by about 30%. On the FC and the COREL data set, both pruning criteria perform similar and are both outperformed by the combination of TP and SSBNN which reduces the costs by $8\% - 15\%$.

This demonstrates that the proposed combination nicely compensates the additional computation costs and that the better approximation of the search space can significantly decrease the amount of page accesses.

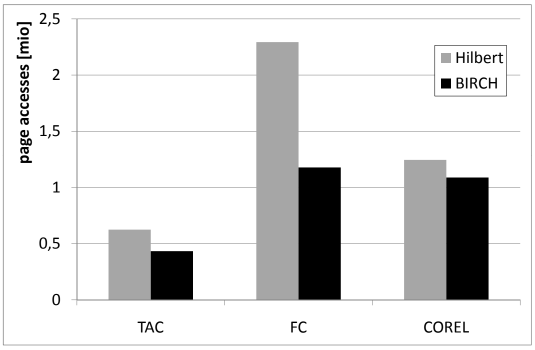


Figure 6.11: Grouping procedures of the outer set, validated via XBNN.

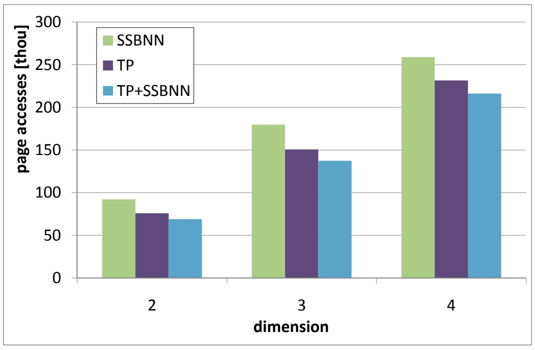


Figure 6.12: A-10-NN queries on clustered data sets of varying dimensions

CPU Consumption

As seen in Sect. 6.4.2, all AkNN-algorithms are clearly I/O bound. However, through the use of buffers and caching strategies, the page accesses can be dramatically reduced. Therefore, it is important to show that TP and TP+SSBNN apply only a negligible computational overhead compared to SSBNN. The results shown in Fig. 6.10 support this claim. Only at the high-dimensional COREL data set, SSBNN is significantly faster with respect to CPU-time. Let us note that also in the presence of buffers, the main challenge for AkNN-algorithms is to reduce the I/O operations. Therefore, it is often beneficial for the overall runtime to reduce I/O operations by the cost of CPU-time.

For all our compared methods it is necessary to group the outer set $\boldsymbol{R}$ into compact query regions. In [176], the authors use a grouping approach based on a bulk-load using the Hilbert order with a maximum threshold for the MBR volume and group size. They argue that this procedure runs in linear time and leads to better page approximations than, for instance, the data pages resulting from indexing $\boldsymbol{R}$ like $\boldsymbol{S}$. This grouping procedure can also be transferred to spherical page approximations. However, we chose BIRCH [178], which is a clustering algorithm with linear runtime. Even though this procedure uses spherical page approximations, we experienced that the groupings resulting from BIRCH show a better suitability even for the AkNN-algorithm based on the X-Tree (XBNN) than those resulting from Hilbert grouping. Fig. 6.11 shows the experiments on the three data sets for the two grouping algorithms. The Hilbert groups have been created by limiting the group volume to the average data page size of the inner set's X-Tree. The parametric setting of BIRCH has been chosen such that the number of groups is comparable to the Hilbert grouping. For the algorithms based on spherical page approximations, the results showed even more evidence for using BIRCH as grouping algorithm.

Synthetic Data Sets

We have also compared SSBNN, TP and TP+SSBNN on several synthetic data sets. Fig. 6.12 displays the results of A-10-NN queries on clustered datasets of varying dimensions. The 3000 clusters in the example data set of size 600,000 have been generated using a Gauss-like process: For each cluster a centroid has been sampled from a uniform distribution. The standard deviation used for generating the cluster points was chosen uniformly for each dimension. The experiments show that the combination of TP and SSBNN outperforms TP by approximately 10% and that it outperforms SSBNN by 20 to 30%. We note that this effect is stable for all dimensions.

6.5 Conclusions

In this chapter, we introduced the *Trigonometric* domination decision criterion which is able to detect $Dom(A, B, R)$ when all three objects are given by spherical approximations. In contrast to previously existing domination criteria it is correct, complete and computable in $O(d)$. To show it's effectiveness we considered the problem of AkNN query processing using an index structure with spherical page approximations (SS-Tree). We developed an AkNN where the domination decision criterion can be easily replaced and compared DDC$_{TR}$ to DDC$_{MM}$ which was adapted to the case of spherical approximations.

Afterwards, we further extend AkNN processing to employ multiple pruning criteria. Thus, it is possible to construct even tighter bounds around the remaining search space and thus to further decrease the number of necessary page accesses. In our experimental evaluation, we demonstrated that our proposed methods decrease the all-over runtime as well as the page accesses necessary to process AkNN queries on three real-world data sets. Especially for larger values for k, our new method considerably improves the query times.

Chapter 7

The Optimal Domination Decision Criterion

The *Trigonometric* domination decision criterion discussed in Chapter 6 provides us with a complete and correct domination decision criterion which is efficiently computable. The drawbacks of the criterion are that it is only applicable for the case where the approximations of the objects are spherical and for the special case of the L_p-Norm where $p = 2$. However the most commonly used approximation for complex objects in the context of databases is still the rectangle. For example MBRs are used as spatial key for the most prominent spatial access methods, e.g. the R-Tree [77], R*-Tree [21], X-Tree [26] as well as specialized adaptations like the TPR tree [137] and the U-Tree [149] among many others. In the last decade, MBR approximations have also become very popular for uncertain databases [35, 52, 49, 110] in order to approximate all possible locations of an uncertain vector object such as a GPS signal.

Thus in this chapter, we propose a novel effective (complete and correct) and efficient criterion to determine the spatial topology between multi-dimensional rectangles. In particular given three rectangles R, A, and B we develop a criterion to determine $Dom(A, B, R)$. We prove that our decision criterion is correct, complete, and efficient to compute even for high dimensional databases. In addition, we tackle the problem of computing the number of objects dominating an object o. The challenge here is to incorporate objects that only partially dominate o. In this work we will show how to detect such partial domination topology by using a modified version of our decision criterion. We propose strategies for conservatively and progressively estimating the total number of objects dominating an object. Our experiments show that when plugged into an application the new domination decision criterion, albeit very general and widely applicable, significantly outperforms current state-of-the-art domination decision criteria. In addition, it is the basis for our novel method to determine conservative and progressive bounds for the domination count of an object efficiently. In particular, we claim the following contributions.

- We propose a novel decision criterion for the domination problem among rectangles that is correct, complete, and can be efficiently computed.

- We propose a number of heuristics that can be used to estimate the domination count in consideration of partial domination.
- We show how our domination decision criterion and our heuristics to determine the domination count can be used to improve spatial pruning strategies for a variety of spatial query processing methods.
- We present extensive experiments to evaluate our new pruning criteria in comparison to state-of-the-art approaches.

Parts of this chapter have been published in [64]. The remainder of this chapter is organized as follows: Section 7.1 introduces our novel domination decision criterion. An effective estimation of the domination count based on this new decision criterion is proposed in Section 7.2. Section 7.3 presents experimental results and Section 7.4 finally concludes the chapter.

7.1 The Criterion

We will derive a new decision criterion that is correct, complete, and can be computed in $O(d)$ time. Our novel domination decision criterion can be derived from the original definition of domination in Definition 3.1 by applying the following six equivalences.

Equivalence 1.

$$\begin{aligned} \forall a \in A, b \in B, r \in R \ &: \ \mathit{dist}(a,r) < \mathit{dist}(b,r) \Leftrightarrow \\ \forall r \in R \ &: \ \mathit{MaxDist}(A,r) < \mathit{MinDist}(B,r) \end{aligned}$$

Proof.
(1) “$\Rightarrow$”
If the left-hand side holds for each $r \in R$ then it also holds for $a \in A$ and $b \in B$ that maximize and minimize the distance to r, respectively. These points $a \in A$ and $b \in B$ obviously determine the values of *MaxDist* and *MinDist*, respectively.

(2) “$\Leftarrow$”
If the right-hand side holds for each $r \in R$ as well as for that $a \in A$ and $b \in B$ that maximizes and minimizes the distance to r, i.e. determines the value of *MaxDist* and *MinDist*, respectively, then it also holds for any $a \in A$ and any $b \in B$. □

Equivalence 2.

$$\begin{aligned} &\forall r \in R : \mathit{MaxDist}(A,r) < \mathit{MinDist}(B,r) \Leftrightarrow \\ &\forall r \in R : \sqrt[p]{\textstyle\sum_{i=1}^{d} \mathit{MaxDist}(A_i,r_i)^p} < \sqrt[p]{\textstyle\sum_{i=1}^{d} \mathit{MinDist}(B_i,r_i)^p} \end{aligned}$$

Proof. Follows directly from the definition of *MaxDist* and *MinDist* for L_p norms (see Section 3.2.2). □

Equivalence 3.
$\forall r \in R : \sqrt[p]{\sum_{i=1}^{d} \mathit{MaxDist}(A_i, r_i)^p} < \sqrt[p]{\sum_{i=1}^{d} \mathit{MinDist}(B_i, r_i)^p} \Leftrightarrow$
$\forall r \in R : \sum_{i=1}^{d} (\mathit{MaxDist}(A_i, r_i)^p - \mathit{MinDist}(B_i, r_i)^p) < 0$

Proof.
$\forall r \in R : \sqrt[p]{\sum_{i=1}^{d} \mathit{MaxDist}(A_i, r_i)^p} < \sqrt[p]{\sum_{i=1}^{d} \mathit{MinDist}(B_i, r_i)^p} \Leftrightarrow$
$\forall r \in R : \sum_{i=1}^{d} \mathit{MaxDist}(A_i, r_i)^p < \sum_{i=1}^{d} \mathit{MinDist}(B_i, r_i)^p \Leftrightarrow$
$\forall r \in R : \sum_{i=1}^{d} \mathit{MaxDist}(A_i, r_i)^p - \sum_{i=1}^{d} \mathit{MinDist}(B_i, r_i)^p < 0 \Leftrightarrow$
$\forall r \in R : \sum_{i=1}^{d} (\mathit{MaxDist}(A_i, r_i)^p - \mathit{MinDist}(B_i, r_i)^p) < 0$ □

Equivalence 4.
$\forall r \in R : \sum_{i=1}^{d} (\mathit{MaxDist}(A_i, r_i)^p - \mathit{MinDist}(B_i, r_i)^p) < 0 \Leftrightarrow$
$\max_{r \in R}(\sum_{i=1}^{d} (\mathit{MaxDist}(A_i, r_i)^p - \mathit{MinDist}(B_i, r_i)^p)) < 0$

Proof. Instead of considering all possible $r \in R$, it is sufficient to consider only that point $r' \in R$ which maximizes the left-hand side of the inequality. If the inequality holds for this point r', then it obviously holds for all possible $r \in R$ and vice versa. □

The next equivalence requires the following lemma:

Lemma 7.1. Let $F : \mathbb{R}^d \to \mathbb{R}$ be a function that is summed by treating each dimension independently, i.e. there exists a function $f : \mathbb{R} \to \mathbb{R}$ such that

$$F(o) = \sum_{i=1}^{d} f(o_i)$$

Also, let $A \subseteq \mathbb{R}^d$ be a rectangle and

$$\sigma := \mathrm{argmax}_{a \in A}(F(a))$$

be the object in A that maximizes F. Then, the following holds:

$$\max_{a \in A}(\sum_{i=1}^{d} f(a_i)) = \sum_{i=1}^{d} \max_{a_i \in A_i}(f(a_i))$$

Proof.

$$\max_{a \in A}(\sum_{i=1}^{d} f(a_i)) \overset{\text{Def } F(a)}{=} \max_{a \in A}(F(a)) \overset{\text{Def } \sigma}{=} F(\sigma)$$

$$\overset{\text{Def } F(a)}{=} \sum_{i=1}^{d} f(\sigma_i) \overset{\text{Def } \sigma}{=} \sum_{i=1}^{d} \max_{a_i \in A_i}(f(a_i))$$

□

Equivalence 5.
$\max\limits_{r \in R}(\sum_{i=1}^{d} \mathit{MaxDist}(A_i, r_i)^p - \mathit{MinDist}(B_i, r_i)^p) < 0 \Leftrightarrow$
$\sum_{i=1}^{d} \max\limits_{r_i \in R_i}(\mathit{MaxDist}(A_i, r_i)^p - \mathit{MinDist}(B_i, r_i)^p) < 0$

Proof. This follows from lemma 7.1 by substituting

$$F(r) = \mathit{MaxDist}(A, r) - \mathit{MinDist}(B, r)$$

□

The final equivalence (equivalence 6) makes the equation computable. It is based on the assumption that for finding the maximum r_i in dimension i, it is sufficient to consider the boundary points (R_i^{min} and R_i^{max}) of the interval R_i. This assumption is proven in the following two lemmas.

Lemma 7.2. Let A and B be intervals. The function $f : \mathbb{R} \to \mathbb{R}$ defined as $f(x) = \mathit{MaxDist}(A, x)^p - \mathit{MinDist}(B, x)^p$ has no local maximum.

Proof. By definition, the value of $MaxDist(A, x)$ only depends on $A.mean$ while the value of $MinDist(B, x)$ only depends on $B.min$ and $B.max$. Since $A.min \geq A.max$, this leads to the three possible cases depicted in Figure 7.1. In the following, we will show for each of the cases that there may not exist any point for which it holds that $f(x)$ is increasing to the left of x and decreasing to the right of x.

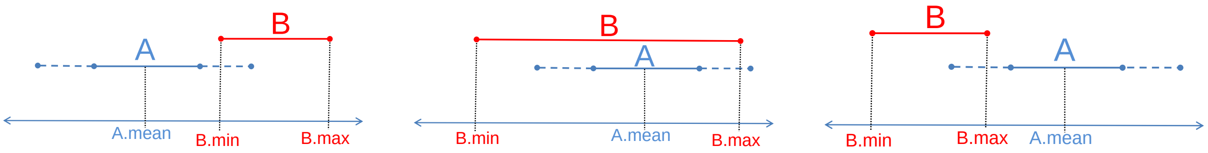
Case 1: $A.mean \leq B.min$ (Figure 7.1(a)):

- For the interval $]-\infty, A.mean]$ Equations 3.1 and 3.2 yield:
  $f(x) = dist(x, A.max)^p - dist(x, B.min)^p =$
  $(|A.max - x|)^p - (|B.min - x|)^p$.
  This is constant if $p = 1$ or if $A.max = B.min$ since both $|A.max - x|$ and $|B.min - x|$ are decreasing. If $p > 1$, this term can be either **monotonically increasing or monotonically decreasing**, depending on whether $A.max$ is greater than $B.min$ or not.

- For the interval $]A.mean, B.min]$ Equations 3.1 and 3.2 yield:
  $f(x) = dist(x, A.min)^p - dist(x, B.min)^p$, where dist(x,A.min) is increasing and dist(x,B.min) is decreasing, thus $f(x)$ is **monotonically increasing** for any p.

- For the interval $]B.min, B.max]$ Equations 3.1 and 3.2 yield:
  $f(x) = dist(x, A.min)^p - 0$.
  This term is **monotonically increasing** since $dist(x, A.min)$ is increasing.

- For the interval $]B.max, \infty[$ Equations 3.1 and 3.2 yield:
  $f(x) = dist(x, A.min)^p - dist(x, B.max)^p =$
  $(|A.min - x|)^p - (|B.max - x|)^p$.
  This is constant for $p = 1$ since both $|A.min - x|$ and $|B.max - x|$ are increasing. Since $A.min < A.mean < B.min < B.max$, above term is **monotonically increasing** for $p > 1$.



(a) $A.mean \leq B.min$ (b) $B.min < A.mean < B.max$ (c) $A.mean \geq B.min$

Figure 7.1: Illustration of Lemma 7.2.

Putting the monotonic pieces together, it is clear that $f(x)$ may have one local minimum at $A.mean$, but definitely has no local maximum.
Case 2: $B.min < A.mean < B.max$ (Figure 7.1(b)):

- For the interval $]-\infty, B.min]$ Equations 3.1 and 3.2 yield: $f(x) = dist(x, A.max)^p - dist(x, B.min)^p = (|A.max - x|)^p - (|B.min - x|)^p$. This term is constant for $p = 1$ since both $|A.max - x|$ and $|B.min - x|$ are decreasing. Since $|A.max - x| \geq |A.mean - x| > |B.min - x|$, above term is **monotonically decreasing** for $p > 1$
- For the interval $]B.min, A.mean]$ Equations 3.1 and 3.2 yield: $f(x) = dist(x, A.max)^p - 0$. This is **monotonically decreasing** for any $p \geq 1$.
- For the interval $]A.mean, B.max]$ Equations 3.1 and 3.2 yield: $f(x) = dist(x, A.min)^p - 0$. This is **monotonically increasing** for any $p \geq 1$.
- For the interval $]B.max, \infty[$ Equations 3.1 and 3.2 yield: $f(x) = dist(x, A.min)^p - dist(x, B.max)^p = (|A.min - x|)^p - (|B.max - x|)^p$.. This is constant for $p = 1$ since both $|A.min - x|$ and $|B.max - x|$ are increasing. Since $|A.min - x| \leq |A.mean - x| < |B.min - x|$, above term is **monotonically increasing** for $p > 1$.

Thus, $f(x)$ has one local minimum at $A.mean$, but definitely no local maximum.

Case 3: $A.mean \geq B.max$ (Figure 7.1(c)): This case is analogous to the first case. □

Lemma 7.3. Let $f : \mathbb{R} \to \mathbb{R}$ be a function that has no local maximum and $I = [I_{min}, I_{max}] \subset \mathbb{R}$ be an arbitrary finite interval. The value that maximizes f in the interval I must be either I_{min} or I_{max}, i.e.

$$\underset{i \in I}{argmax}(f(i)) \in \{I_{min}, I_{max}\}$$

Proof. Let $p \in [I_{start}, I_{end}]$ be the value that maximizes f in I, i.e. $p = \text{argmax}_{i \in I}(f(a))$. Then, $\forall i \in I : f(i) \leq f(p)$, in particular, $f(I_{min}) \leq f(p)$ and $f(I_{max}) \leq f(p)$. Note that $f(I_{min}) < p$ and $f(I_{max}) < p$ cannot both be true, because this would be a contradiction to the assumption that $f(x)$ has no local maximum. Thus it must either hold that $f(I_{start}) = f(p)$ or $f(I_{end}) = f(p)$, i.e. $I_{min} = \text{argmax}_{i \in I}(f(x))$ or $I_{max} = \text{argmax}_{i \in I}(f(x))$. □

Now we can derive the final equivalence.

Equivalence 6.
$\sum_{i=1}^{d} \max_{r_i \in R_i}(\mathit{MaxDist}(A_i, r_i)^p - \mathit{MinDist}(B_i, r_i)^p) < 0 \Leftrightarrow$

$$\sum_{i=1}^{d} \max_{r_i \in \{R_i^{min}, R_i^{max}\}}(\mathit{MaxDist}(A_i, r_i)^p - \mathit{MinDist}(B_i, r_i)^p) < 0$$

Proof. Follows from lemma 7.2 and 7.3. □

Definition 7.1 (Optimal Domination Decision Criterion)**.** Our novel optimal domination decision criterion is defined as

$$\mathrm{DDC}_{OPT}(A, B, R) \Leftrightarrow \sum_{i=1}^{d} \max_{r_i \in \{R_i^{min}, R_i^{max}\}}(\mathit{MaxDist}(A_i, r_i)^p - \mathit{MinDist}(B_i, r_i)^p) < 0$$

Lemma 7.4. The novel optimal domination decision criterion is correct and complete.

Proof. Correctness and completeness follow directly from equivalences 1 to 6. □

Obviously, the novel optimal domination decision criterion can be computed in $O(d)$ time and, thus, fulfills all three desired properties mentioned in Section 3.1.

Using our novel domination decision criterion DDC_{OPT}, the basic domination count DC_{basic} can be computed in $O(N \cdot d)$. This is worth noting since existing decision criteria only allow either to compute the exact DC_{basic} value in exponential time or to compute an approximation of the DC_{basic} value in linear time. In the latter case, we would obtain a lower bound of DC_{basic} which makes the lower bounding estimation of the domination count even more loose.

7.2 Domination Count Computing

The computation of the domination count $DC(\mathcal{O}, B, R)$ of an object B w.r.t. object R can be separated into two sub problems:

- First, compute the number of rectangles that (completely) dominate the rectangle B w.r.t. rectangle R. This can be done by computing DC_{basic} (cf. Definition 3.5) using DDC_{OPT}.
- Second, take into account all sets of rectangles that increase the domination count of B as a group and that do not contain any element that does so separately. For this part we need the concept of *partial domination* (cf. Definition 3.6).

Obviously, the sets of objects that dominate B as a group can only contain rectangles $A_i \in \mathcal{O}$ that partially dominate B, i.e. for which $PDom(A_i, B, R)$ holds. In other words, for the computation of the second part of the domination count of a rectangle B, we could use the detection of partial domination as a first step because only those rectangles A_i for which $PDom(A_i, B, R)$ holds could be the elements of those set of rectangles that dominate B as a group.

In the following we will discuss how our novel domination decision criterion DDC$_{OPT}$ can be used for (i) detecting partial domination in $O(d)$ and (ii) deriving a conservative approximation of the domination count.

7.2.1 Partial Domination

Partial domination can efficiently be detected by applying the following six equivalences analogously to Section 7.1. We start with inequality 3.4.

Equivalence 7.

$$\begin{aligned}\exists r \in R : \forall a \in A, b \in B &: \mathit{dist}(b,r) > \mathit{dist}(a,r)\\ \Leftrightarrow \exists r \in R &: \mathit{MaxDist}(A,r) < \mathit{MinDist}(B,r)\end{aligned}$$

Proof. This proof is analogous to the proof of Equivalence 1, i.e. it exploits that DDC$_{MM}$is optimal in the case where R is a point. □

Equivalence 8.

$$\exists r \in R : \mathit{MaxDist}(A,r) < \mathit{MinDist}(B,r) \Leftrightarrow$$
$$\exists r \in R : \sqrt[p]{\sum_{i=1}^{d} \mathit{MaxDist}(A_i,r_i)^p} < \sqrt[p]{\sum_{i=1}^{d} \mathit{MinDist}(B_i,r_i)^p}$$

Proof. Follows directly from the definition of *MaxDist* and *MinDist* for L_p norms. □

Equivalence 9.

$$\exists r \in R : \sqrt[p]{\sum_{i=1}^{d} \mathit{MaxDist}(A_i,r_i)^p} < \sqrt[p]{\sum_{i=1}^{d} \mathit{MinDist}(B_i,r_i)^p} \Leftrightarrow$$
$$\exists r \in R : \sum_{i=1}^{d} \mathit{MaxDist}(A_i,r_i)^p - \mathit{MinDist}(B_i,r_i)^p < 0$$

Proof.

$$\exists r \in R : \sqrt[p]{\sum_{i=1}^{d} \mathit{MaxDist}(A_i,r_i)^p} < \sqrt[p]{\sum_{i=1}^{d} \mathit{MinDist}(B_i,r_i)^p} \Leftrightarrow$$
$$\exists r \in R : \sum_{i=1}^{d} \mathit{MaxDist}(A_i,r_i)^p < \sum_{i=1}^{d} \mathit{MinDist}(B_i,r_i)^p \Leftrightarrow$$
$$\exists r \in R : \sum_{i=1}^{d} \mathit{MaxDist}(A_i,r_i)^p - \sum_{i=1}^{d} \mathit{MinDist}(B_i,r_i)^p < 0 \Leftrightarrow$$
$$\exists r \in R : \sum_{i=1}^{d} \mathit{MaxDist}(A_i,r_i)^p - \mathit{MinDist}(B_i,r_i)^p < 0$$

□

Equivalence 10.

$$\exists r \in R : \sum_{i=1}^{d} \mathit{MaxDist}(A_i,r_i)^p - \mathit{MinDist}(B_i,r_i)^p < 0 \Leftrightarrow$$
$$\min_{r \in R}\left(\sum_{i=1}^{d} \mathit{MaxDist}(A_i,r_i)^p - \mathit{MinDist}(B_i,r_i)^p\right) < 0$$

Proof. The rationale for equivalence 10 is that if there exists an $r \in R$ for which the left-hand side returns less than 0, then this also holds for the r which minimizes the term on the right-hand side and vice versa. □

Equivalence 11.
$\min_{r\in R}(\sum_{i=1}^{d} \mathit{MaxDist}(A_i, r_i)^p - \mathit{MinDist}(B_i, r_i)^p) < 0 \Leftrightarrow$
$\sum_{i=1}^{d} \min_{r_i\in R_i}(\mathit{MaxDist}(A_i, r_i)^p - \mathit{MinDist}(B_i, r_i)^p) \ < \ 0$

Proof. This proof is analogous to the proof of Equation 5 using minimization instead of maximization. □

Analogously to Equivalence 6, the last equivalence below makes the equation computable. Again, we need two lemmas.

Lemma 7.5. Let D be a one dimensional vector database using L_p-Norm. Let A and B be intervals. The function $f : \mathbb{R} \to \mathbb{R}$:

$$f(x) = \mathit{maxDist}(A, x)^p - \mathit{minDist}(B, x)^p$$

may have a local minimum only at $A.mean$.

Proof. This proof is contained in the formal proof of lemma 7.2. □

Lemma 7.6. Let $f : \mathbb{R} \to \mathbb{R}$ be a function that has at most one local minimum at x. For any finite interval $I \subset \mathbb{R} = [I_{start}, I_{end}]$ the following holds:

$$\underset{i\in I}{argmin}(f(i)) \in \{I_{start}, I_{end}, x\}$$

That is, the point of the interval I that minimizes $f(x)$ must be either the lower or the upper bound of I, or the local minimum x.

Proof. The proof is similar to the proof of Lemma 7.3 and thus omitted here. □

In consideration of the above lemmas we now derive the final equivalence:

Equivalence 12.
$\sum_{i=1}^{d} \min_{r_i\in R_i}(\mathit{MaxDist}(A_i, r_i)^p - \mathit{MinDist}(B_i, r_i)^p) < 0 \Leftrightarrow$
$\sum_{i=1}^{d} min_{r_i}(\mathit{MaxDist}(A_i, r_i)^p - \mathit{MinDist}(B_i, r_i)^p) < 0,$
$\quad where\ r_i \in \{R_i^{min}, R_i^{max}, A_i^{mid}\}$

Proof. Directly follows from Lemma 7.5 and Lemma 7.6. □

Thus, using the formula in Equivalence 12 we can efficiently detect all partial dominations. However, as indicated above, this is only the first step towards computing the domination count. In fact, we need to determine that subregion of the reference rectangle R, for which the domination count is minimal. Since we cannot test all possible points $r \in R$ (see also the discussion above), we propose three heuristics to conservatively approximate the domination count of a rectangle.

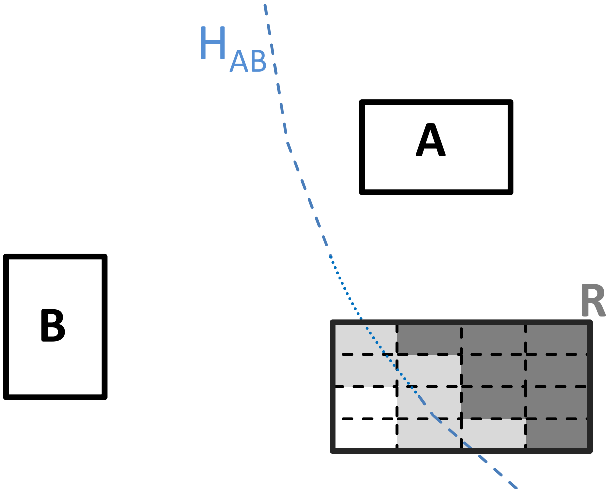


Figure 7.2: Partial domination using grid partitioning

7.2.2 Domination Count Estimation

Using the techniques proposed in Sections 7.1 and 7.2.1 we can check if an MBR A dominates B completely or partially w.r.t. R. These tests are generally applicable as long as the involved objects are MBRs. For calculating the domination count of B it is therefore possible to split R into smaller MBRs and then calculate the domination count for each cell individually. The following three heuristics use different approaches for splitting R to estimate the domination count.

Domination Count Estimation based on grid partitioning

A straight forward approach for splitting R is performed by using a grid with a fixed number m of partitions in each dimension. Considering the example in Figure 7.2, we can (using the decision criteria for domination and partial domination) assert that A dominates B w.r.t. all dark gray cells and partially dominates B w.r.t. all light gray cells of R. For the rest of the cells (white) A does not dominate B. Using this grid partitioning, the domination count ($DC(\mathcal{O}, B, R)$) can be estimated by the minimum domination count of all cells $c_i \in R$, that is:

$$DC_{grid}(\mathcal{O}, B, R) = min_i(DC_{basic}(\mathcal{O}, B, c_i))$$

This estimation is valid as we know that B is dominated by at least this amount of $A_i \subset \mathcal{O}$ w.r.t. each cell $c_i \in R$.

An example for the grid based partial pruning is given in Figure 7.3. Here an MBR R is partitioned into 16 cells. In addition two Voronoi hyperplanes H_{A_1B} and H_{A_2B} are shown. The objects $\mathcal{O} = \{A_1, A_2\}$ and B generating the hyperplanes are ommited here. For the area on the right-hand side of H_{A_1B}, object B is dominated by object A_1 and for the left-hand side of H_{A_2B}, B is dominated by A_2. It is clear that neither A_1 nor A_2 (fully-)

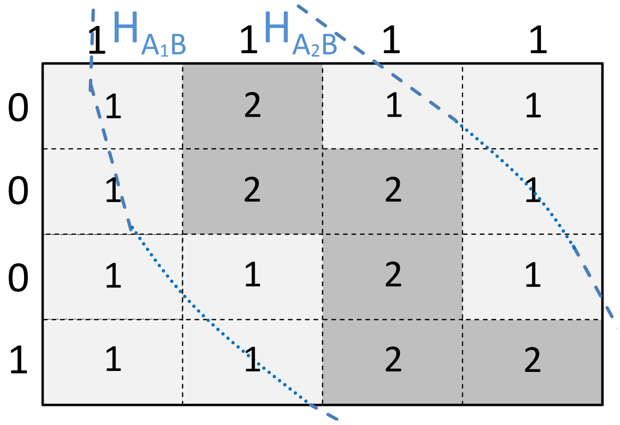


Figure 7.3: Domination Count estimation using grid partitioning.

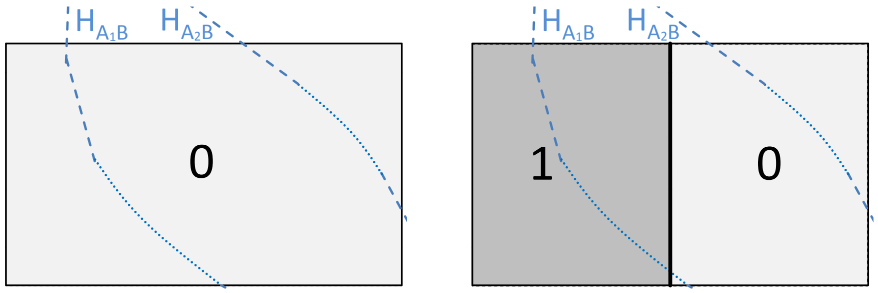
dominate B with respect to the whole MBR R. For each cell the conservative domination count $DC_{basic}(\mathcal{O}, B, c_i)$ is shown. With respect to dark cells, A_1 and A_2 dominate B and thus the cells have a value of 2. With respect to light cells, only one of the two objects dominates B, therefore they get marked with a value of 1. By taking the minimum value of all cells $c_i \in R$ we obtain $DC_{grid}(\mathcal{O}, B, R) = 1$. The advantage of this approach is, that it returns a very accurate estimation of the domination count while avoiding expensive materialization of the Voronoi hyperplanes. The accuracy can be boosted by increasing the number of splits per dimension. In return increasing m will highly increase the runtime of the algorithm, as the number of cells $c_i \in R$ is m^d. This implies that this approach is not applicable for high dimensions. For each cell c_i, $DC_{basic}(\mathcal{O}, B, c_i)$ can be computed in a single scan of the objects for which $PDom(A_i, B, R)$ holds using the DDC$_{OPT}$ (c.f. Definition 7.1). Thus the total time complexity is in $O(d \cdot |\mathcal{O}| \cdot m^d)$.

Domination Count Estimation based on slices

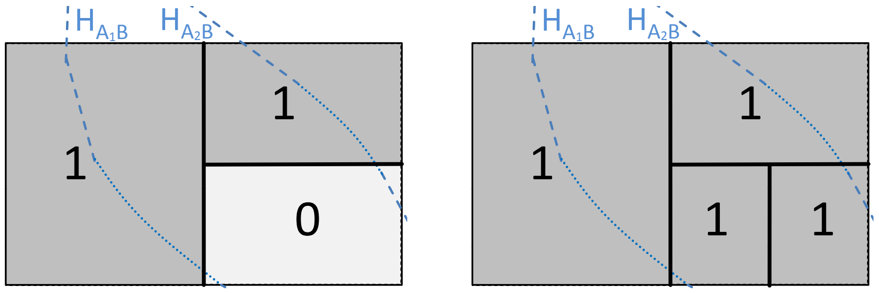
In order to reduce the runtime of the domination count estimation, we propose a second algorithm, which is not based on a grid partitioning. Instead of cells, this approach considers *slices*. Therefore an MBR R is split into m *slices* s_i^{dim} in each of the d-dimensions $(1 \leq dim \leq d)$. This results in $d \cdot m$ overlapping *slices*. The domination count $DC(\mathcal{O}, B, R)$ can then be approximated by computing, for each dimension, the minimal domination count of all slices and using the result of the dimension maximizing this estimation.

$$DC_{slice}(\mathcal{O}, B, R) = max_{dim}(min_i(DC_{Basic}(\mathcal{O}, B, s_i^{dim})))$$

For example, the domination count $DC(\mathcal{O}, B, s_i)$ for each slice s_i (i.e. each row and each column) and each cell c_i is shown in Figure 7.3 for a 2 dimensional MBR R. The minimal domination count considering all rows is 0, while the minimal domination count w.r.t. all columns is 1. Thus $DC_{slice}(\mathcal{O}, B, s_i) = 1$ in this example. The complexity of this algorithm is in $O(m \cdot d)$. However, this approach yields much worse results than the



(a) initial setting (b) after 1st split



(c) after 2nd split (d) after 3rd split

Figure 7.4: Example for computing DC_{bisect}

grid-based approach for an identical m parameter. Details can be found in our experiments (Section 7.3).

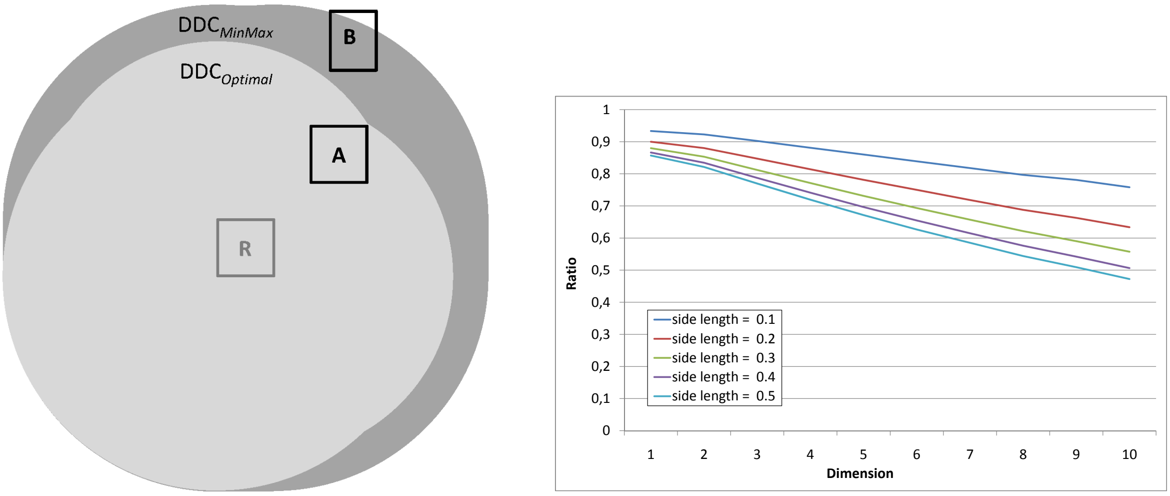
Domination Count Estimation based on bisections

We next propose a bisection based approach that yields much better efficiency, while still being linear in d. This approach works iteratively. During each iteration, one section of R is chosen to be split evenly (mean split) in one dimension. After m splits, this results in $m+1$ sections $s_0 \cup s_1 \cup \ldots \cup s_m = R$ and it holds that:

$$DC_{bisect}(\mathcal{O}, B, R) = min_i(DC_{basic}(\mathcal{O}, B, s_i))$$

The challenge here is to wisely choose the split section of R and the dimension to split in each iteration.

We propose to split the section $s \subset R$ with the lowest domination count estimation. This decision is optimal, because the estimation of $DC(\mathcal{O}, B, R)$ is determined by the section which results in the lowest domination count. Thus, in order to increase the domination count approximation, s must be split. If the decision for s is ambiguous, then one of the candidates of s is chosen arbitrarily. To determine the split axis, the heuristic tests each dimension, and greedily uses the dimension that yields the highest domination count $DC_{bisect}(\mathcal{O}, B, R)$ considering the two resulting bisections of s. In the case of ties the axis is chosen which maximizes the sum $\sum_{i=0}^{m} DC_{basic}(\mathcal{O}, B, s_i)$. An example is shown in



(a) Refinement areas for fixed R and A

(b) Ratio of the refinement areas of DDC$_{OPT}$ and DDC$_{MM}$ w.r.t. dimension and size of MBRs

Figure 7.5: Refinement areas of DDC$_{MM}$ and DDC$_{OPT}$

Figure 7.4. Considering Figure 7.4(a) it is clear that none of the two objects $A_1, A_2 \in \mathcal{O}$ that are responsible for the Voronoi hyperplanes H_{A_1B} and H_{A_2B} dominates B w.r.t. R. Beginning with the y-axis as split axis would result in two equi-sized MBRs both of which result in a domination count $DC_{bisect}(\mathcal{O}, B, R)$ of 0 and therefore the approximation of $DC(\mathcal{O}, B, R)$ does not increase. Choosing the x-axis as split axis would result in two equi-sized MBRs shown in Figure 7.4(b) yielding the same domination count approximation but a higher sum ($\sum_{i=0}^{m} DC_{basic}(\mathcal{O}, B, s_i) = 1$). In the next iteration, the right MBR is chosen to be split, since it is responsible for the lowest domination count approximation. Both possible split axes are equal according to our heuristic. In the example, the y-axis is chosen arbitrarily (c.f. Figure 7.4(c)). The third (see Figure 7.4(d)) split of the lower-right MBR increases $DC_{bisect}(\mathcal{O}, B, R)$ to 1.

The bisection-based Domination Count Estimation algorithm uses m iterations. In each iteration i there exist exactly i sections of which the section with the lowest conservative domination count has to be found. This yields a complexity of $O(m^2)$ but can be reduced to $O(m \cdot log(m)))$ by using a Priority Queue to find the section with the lowest conservative domination count. For the greedy heuristic, in each iteration, each dimension has to be tested to determine the best split axis in $O(m \cdot d)$. Thus we get a total complexity of $O(m \cdot log(m) + m \cdot d) = O(m \cdot \max(log(m), d))$, where m is the number of iterations.

7.3 Experimental Evaluation

This section evaluates the effectiveness and efficiency of our novel domination decision criterion in comparison to the prevalent DDC_{MM} decision criterion [1]. After that we evaluate the performance of the domination count estimation approaches from Section 7.2.2. Finally, we will exemplarily show how our new methods influences the performance of existing similarity search method designed for kNN and RkNN queries. For all experiments the underlying distance function is the Euclidian norm.

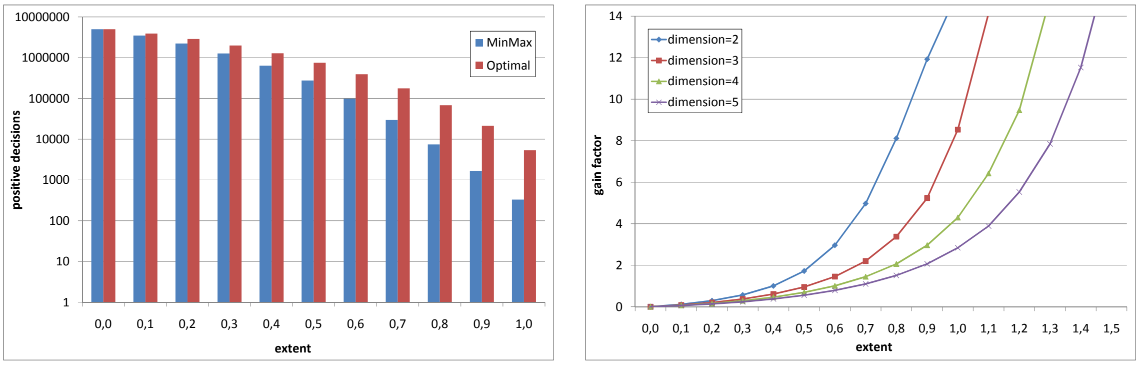
7.3.1 Single Object Domination

We first evaluate the effectiveness gain of DDC_{OPT} compared to DDC_{MM} in consideration of the decision power. In order to measure the decision power, we take for a given pair of rectangles R and A the region into account containing all points that cannot be detected to be dominated by A w.r.t. R. In the reminder we call this region refinement area, since all objects intersecting this area might be refined in order to detect the domination relation. It should be clear that the smaller this area, the higher the corresponding domination power. Figure 7.5(a) exemplarily shows the refinement areas for the 2-dimensional MBRs A and R w.r.t. both criteria DDC_{MM} and DDC_{OPT}, respectively. In this example, object B is detected to be dominated by A only if we apply DDC_{OPT} instead of DDC_{MM}. The refinement areas depend on several conditions such as position, shape, distance and extension of the MBRs specifying the refinement area as well as the dimensionality of the space. For our experiment evaluating the domination power, pairs of MBRs R and A are positioned in $[0,1]^d$ with a fixed *MinDist* of 0.5 and equal distances in each dimension. The length of each side of the two MBRs was scaled from 0.1 to 0.5 and dimension screened from 1 to 10. The gain of the domination power is measured by the ratio of the volumes of the refinement area w.r.t. DDC_{OPT} and the refinement area w.r.t. DDC_{MM} by means of Monte-Carlo-Sampling. The results in Figure 7.5(b) show that DDC_{OPT} leads to a much higher decision power. The effect becomes more evident as the number of dimensions and the extension of the MBRs increase. As expected, increasing the extension of the MBRs leads to diminishing completeness of the DDC_{MM} decision criterion. It is notable, that the DDC_{MM} criterion suffers considerably from an increasing dimensionality. Note that we used MBRs of equal side length as we observed that this setting favors the decisions power based on DDC_{MM} in order to make a fair comparison. In fact, the advantage of the gain of the decision power based on DDC_{OPT} will increase even further for non-quadratic rectangles.

In addition to the above experiment which is more from a theoretical point of view, we compared the number of domination relations detected by applying DDC_{OPT} and DDC_{MM} . Therefore we randomly generated one million triples of rectangles (A, B, R) with a fixed extent (i.e. the sum of side lengths) in the $[0,1]^2$ space. For each triple

---

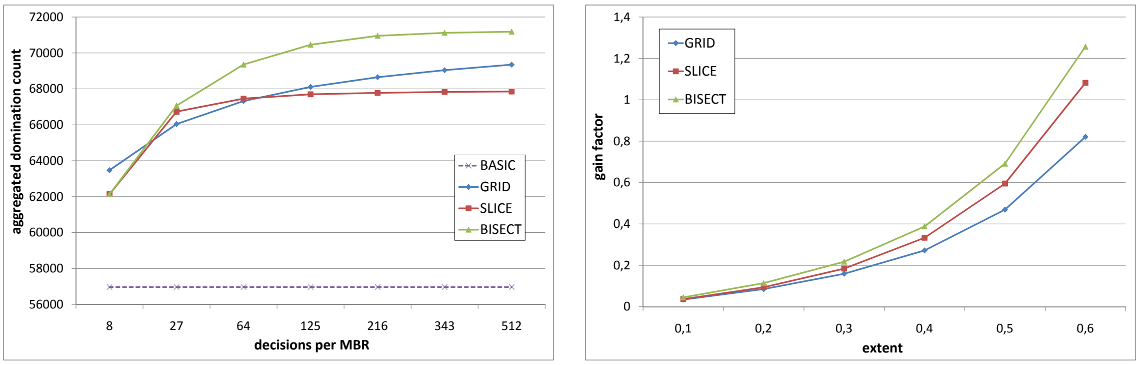
[1] Let us note, that we did not compare against DDC_{CB} since it yields the same effectivity than DDC_{OPT} at the price of higher computational cost



(a) Positive decisions made

(b) Factor of positive decisions made more by the optimal criterion

Figure 7.6: Comparison of MinMax- and optimal-criterion on synthetic data



(a) Accuracy vs. efficiency

(b) Performance w.r.t. MBR size

Figure 7.7: Heuristics for partial domination

we tested if the decision criterion is able to determine whether $Dom(A, B, R)$ holds. Finally we aggregated the number of positive decisions for different extents of the MBRs. The results are illustrated in Figure 7.6(a). Note that an extent of zero yields points instead of rectangles such that both criteria perform equal. However, we can observe that with increasing extent, the percentage of positive decisions of DDC_{OPT} compared to DDC_{MM} increases considerably. The gain of the decision power based on DDC_{OPT} over DDC_{MM} is illustrated in Figure 7.6(b) showing the factor of positive domination decisions using DDC_{OPT} in comparison of that using DDC_{MM}. We varied the dimensionality of the rectangle space up to 5 dimensions. Here we can observe that the gain increases with increasing extent. In contrast, when increasing the dimensionality, the gain of the decision power decreases. The reason is that in this setting, the extent of the MBRs is fixed for all dimensionality settings such that the average side length per dimension decreases and MBRs converge to points for high dimensionality.

7.3.2 Domination Count Estimation

The next experiments evaluate the accuracy of the domination count estimation of a rectangle B w.r.t. a rectangle R for the approaches proposed in Section 7.2.2: Basic Domination Count Estimation (DC_{basic}), grid partitioning (DC_{grid}), slice partitioning (DC_{slice}) and bisection based partitioning (DC_{bisect}). For these experiments, we generated one thousand three-dimensional MBRs with random position. One MBR R was positioned in the center of the data space. Then we computed the conservative domination count w.r.t. R for each MBR using the four approaches mentioned above. We performed several runs for different parametric settings and averaged the results. Figure 7.7 shows the performance of all four approaches in terms of estimated domination count. First, we want to get a grasp of the relationship between accuracy of the domination count estimation and the cost required for the domination count computation. Therefore, the cost is measured in terms of number of calls of DDC$_{OPT}$. It should be clear that when increasing the number of MBR partitions, and thus the required number of DDC$_{OPT}$ calls, the estimation accuracy of all approaches improves, except for the basic approach since it does not use any partitioning. Figure 7.7(a) shows the results for MBRs with an extent of 0.3. It can be seen that all approaches show a significant improvement compared to DC_{basic} when increasing the number of allowed DDC$_{OPT}$ calls. In particular, the accuracy increases very fast at the beginning of the partitioning process but slows down later on. We can also observe that DC_{bisect} significantly outperforms the other approaches when allowing more than 27 DDC$_{OPT}$ calls per MBR, while DC_{grid} performs best for 8 or less DDC$_{OPT}$ calls.

In the next experiment, as shown in Figure 7.7(b), we fixed the number DDC$_{OPT}$ calls per MBR to 64 and varied their extent. We measured the gain of the domination count over DC_{basic}. Here, again, DC_{bisect} outperforms the other approaches in particular for larger MBR sizes. Note that for a given application, the optimal number of partitions depends on the cost for evaluating a candidate object. The higher that cost, the more partitions can be used in order to reduce the total runtime.

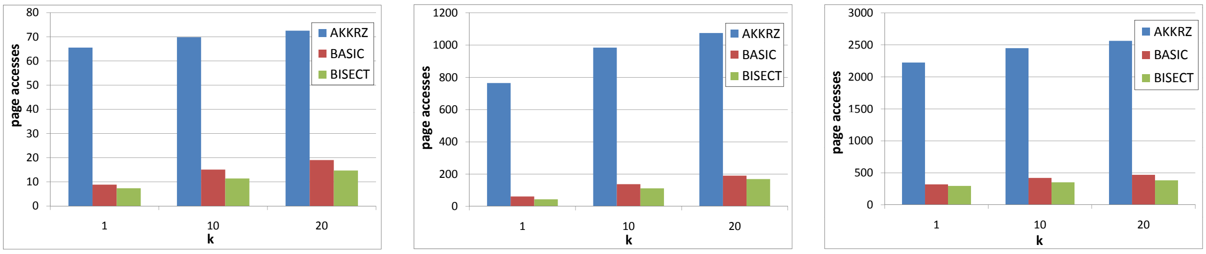
7.3.3 Impact on Standard Spatial Query Processing Methods

In our last experiments, we evaluate the impact of our approaches on the performance of standard query processing methods. Here, we refer to Section 3.3.2, describing how our methods can be plugged into state-of-the-art query processing methods. In particular, we exemplarily consider the most prominent query methods, the k-nearest neighbor search and the reverse k-nearest neighbor search. For this evaluation we use one synthetic dataset, containing 100k uniformly distributed 5D points, and two real world datasets *TAC*[175] consisting of 705099 2D points and *Forest*[80] containing 581012 10D points.

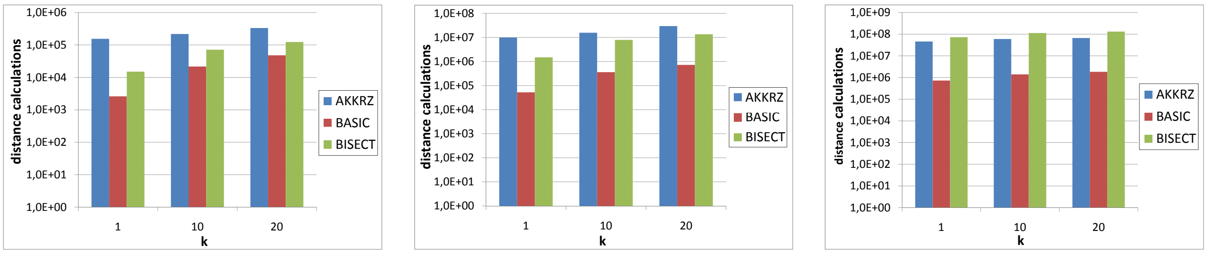
First, we evaluate the impact of our two domination-count estimation approaches DC_{basic} and DC_{bisect} on a reverse k-nearest neighbor search method. As a baseline, we use the algorithm proposed in [4] (in the following referred to *AKKRZ*) for RkNN search on the Euclidean space using an R^*-Tree[2]. The *AKKRZ* algorithm originally uses the Min-

---

[2] We use the R^*-Tree provided in the Elki Framework [5]



(a) TAC dataset (b) $Uniform$ dataset (c) $Forest$ dataset

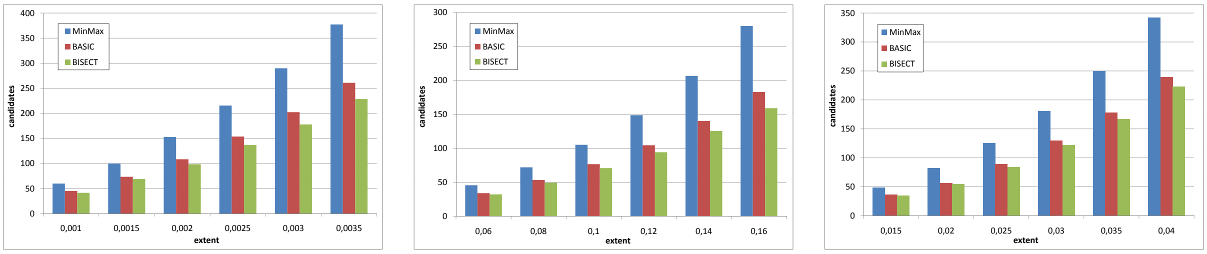


(d) TAC dataset (e) $Uniform$ dataset (f) $Forest$ dataset

Figure 7.8: *AKKRZ* using different decision criteria. Page accesses (top row) and distance calculations (bottom row).

MaxDist domination decision criterion to conservatively prune candidates. We evaluate the impact by comparing the query performance of the original AKKRZ algorithm with the version where we replace the domination count estimation with our methods. Note, that with except of the domination count estimation method, both RkNN versions are identical. The results illustrated in Figures 7.8(a), 7.8(b) and 7.8(c) show the query performance of both RkNN versions in terms of average number of page accesses for varying parameter k and different datasets. It is notable that the enhanced algorithm requires less page access by almost a full order of magnitude on all datasets. Using DC_{bisect} to apply the paradigm of partial pruning based on bisections (c.f. Section 7.2.2) with a maximum number of ten splits per MBR, the number of page accesses can be decreased even further. The large performance increase compared to the original version of *AKKRZ* can be explained by the fact that our domination decision criterion has a much higher pruning power on large MBRs compared to the original version that is based on the MinMaxDist domination decision criterion. This allows us to prune candidates already on a high directory level and, thus, to prune a large number of candidate MBRs very early.

Beside the I/O cost, it is also important to consider the CPU cost since the accuracy of our domination count estimation methods is highly influenced by the CPU cost spent for the estimation process, as shown in the previous section. For this reason, in addition to the I/O cost evaluation we evaluate the CPU cost measured by the number of distance calculations required for the competing techniques as the CPU cost are mainly distance computation bounded. We counted the total number of distance calculations. Calls of



(a) TAC dataset (2D) (b) $Uniform$ dataset (5D) (c) $Forest$ dataset (10D)

Figure 7.9: Evaluation of the different decision criteria for 10 nearest neighbor queries.

DDC$_{OPT}$ and DDC$_{MM}$ were penalized with two distance calculations[3]. The resulting numbers of total distance calculations are shown in Figures 7.8(d), 7.8(e) and 7.8(f). It can be observed that the enhanced $AKKRZ$ algorithm using DC_{basic} significantly outperforms the basic $AKKRZ$ by close to two orders of magnitude. The rationale for this is that the number of calculations increases quadratic in the number of candidates. However, the high computational cost required when applying partial pruning becomes evident here. Using DC_{bisect} with a maximum of ten splits, the number of distance calculations increases by a factor of about five.

Finally, we evaluate the impact of DDC$_{OPT}$ and partial domination on kNN queries among objects approximated by MBRs. These experiments are based on three artificial datasets that rely on the three datasets used in the foregoing experiments (TAC, Uniform, Forest). Each vector in a dataset defines the center of an MBR. For each of the resulting datasets 100 MBRs were chosen randomly as query MBR Q for a 10-NN query on the remaining dataset. Here we did not apply any index structure. The performance of the competing approaches were measured by the average number of candidates that could neither be pruned nor be reported as true hits. The results showing the performance in terms of the number of remaining candidates are depicted in Figure 7.9 for varying extent of the MBRs. It can be observed, that DC_{basic} significantly reduces the number of candidates compared to pruning based on DDC$_{MM}$ on all datasets. The relative performance boost increases for an increasing extent of the MBRs. We also found out in our experiments, that the parameter k has no significant influence on the relative performance boost. Figure 7.9 also shows, that DC_{bisect} is able to further boost the performance, especially for large MBRs independent of the dataset.

7.4 Conclusions

As chapters 3 - 7 revealed the concept of spatial domination is a very useful tool to perform spatial pruning on rectangles and other approximations in a wide field of applications. The domination decision criteria can be plugged into several different query types like reverse

---

[3] in concordance with run-time experiments

nearest neighbour queries, reverse nearest neighbour ranking and all-k-nearest neighbour queries. Since existing domination decision criteria were either incomplete (*MinMaxDist* domination decision criterion) or were subject to some crucial restrictions (*Hyperplane* domination decision criterion) we successively developed more powerful criteria removing these constraints. Starting from the *Hyperplane* criterion we developed the *EntryHyperplane* where approximation A in the relation $Dom(A, B, R)$ was allowed to be extended. With the gained insight we were able to construct the *CornerBased* domination decision criterion which also allowed approximation B to be extended, still yielding exponential runtime in the number of dimensions. The *Trigonometric* criterion removed this drawback by introducing spherical approximations. In this last chapter of Part II of this work we proposed a domination decision criterion that is complete, correct and efficiently computable in $O(d)$ on rectangular approximations. In addition, we discussed how this decision criterion can be used to accurately estimate the domination count of objects by incorporating information about partial domination. While all current approaches that use information about partial domination can only be used on low dimensional data our solution can be applied to data of arbitrary dimensionality. Our experimental evaluation shows that our novel decision criterion can be used to vastly increase the pruning power of existing applications.

Part III

Distance Dependencies in Uncertain Spatial Data

Human science is an uncertain guess.
Edward G. Prior

Chapter 8

Uncertain Data

8.1 Introduction

In recent years *uncertain data* has emerged as a hot topic in the database research community. This trend can be explained by the fact that there is a immanent need for a correct handling of uncertainty in many applications collecting and storing data. Examples how uncertain data can arise in a database are:

- Automatic Knowledge Extraction: When natural language processing techniques are used to extract knowledge from unstructured web documents, the extracted information will usually not be 100% accurate. Using uncertain data modelling it is possible to assign probabilities corresponding to the level of confidence with the extracted information. For example we may assign a probability of 0.7 to the fact that "Buck works at the LMU".
- Human Readings: A similar issue may arise with human observations. Suppose a witness saw a red car, leaving the scene of an accident but is not sure whether the brand of the car was a "Toyota" (probability of 0.75) or a "Mazda" (probability of 0.25).
- Sensor Readings: Sensors measuring environmental variables are never fully accurate due to discretization or inherent uncertainty. Thus a sensor may only report that the temperature is $30°C \pm 2°C$ and GPS devices are only able to indicate the current position with several meters of accuracy.
- Extrapolated Data: Extrapolated data, e.g. weather forecasts are inherently uncertain. For example the chance of sunny weather next Tuesday in Munich is only 30%.

These examples illustrate that data in many applications is never fully certain, thus dealing with this uncertainty is an important and highly relevant problem. One approach to do this is to clean the data in order to remove the uncertainty and then store the data in a traditional DBMS. However this cleaning process is not trivial and requires a lot of effort dependent on the data and the application [134]. Another approach is to keep the uncertainty in the data and treat the values "correctly" during query processing meaning assigning probabilities to query results. This approach does not require any preprocessing of the data but query processing becomes much more challenging. The second approach however has two huge advantages:

- Each result is now assigned with a confidence (probability) and thus lets the user much better decide how to interpret the outcome of a query. In contrast after data cleaning it is not clear to the user how the data has been changed and reorganized.
- Relationships between different entries in the database remain existent and can be interpreted during query processing according to the understanding of the user.

Following these thoughts, this part will focus on the correct management and handling of uncertain data specifically with the goal to support efficient similarity search on this data. Parts of this chapter have been published in [30].

8.2 Modelling Uncertain Data

The management of uncertain data has gained increasing interest in diverse application fields, e.g. sensor monitoring [54], traffic analysis, location-based services [167] etc. Thus the topic of uncertain data, i.e., data containing potentially uncertain information, has received a lot of attention from the database research community in recent years [10, 143, 145]. It involves a number of challenges in terms of collecting, modelling, as well as indexing and querying uncertain data. Though uncertain data management has been studied in the traditional database literature [16, 102, 18, 42, 69, 73], this field has seen a revival, recently, due to new techniques for collecting inherently uncertain data. As a consequence, novel concepts for managing uncertain data have been developed. Most prominent concepts for probabilistic data management are MayBMS [10], MystiQ [37], Trio [6] and MCDB [88]. These uncertain database management systems (UDBMS) provide solutions to cope with uncertain relational data. In the following we will first focus on the modelling of uncertain data in relational databases in general and afterwards consider the special case of uncertain spatial databases.

8.2.1 Categorization

According to [142] uncertainty in relational databases can be categorized in *tuple uncertainty* and/or *attribute uncertainty.*

Following the *tuple uncertainty* model, each tuple in the database is associated with a probability that the tuple is existent. This model is often also called *existential uncertainty.* An example where the incorporation of uncertainty as *tuple uncertainty* is used is shown in Table 8.1. Here the data is collected by an indoor RFID tracking system. RFID tracking

ID	**Time**	**Person**	**Location**	**Prob.**
t1	Jack	10:15	Jack's Office	0.8
t2	Diane	10:20	Conference Room	1.0
t3	Jack	10:20	Kitchen	0.4
t4	Jack	10:25	Conference Room	0.9

Table 8.1: Tuple uncertainty example

ID	**Witness**	**Time**	**Car**	**Color**
t1	John	11:30	[Mazda : 0.2] $\oplus$ [Toyota : 0.8]	red
t2	Adam	11:35	[Honda : 0.6] $\oplus$ [Mazda : 0.4]	red
t3	Bob	11:30	Mazda	[red : 0.7] $\oplus$ [orange : 0.3]

Table 8.2: Attribute uncertainty example

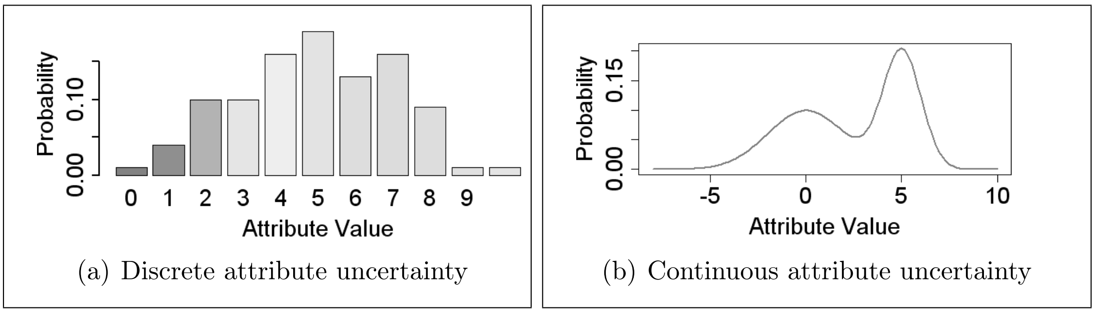
is however subject to several sources of error and is thus not always accurate [135]. This leads to data tuples in the database which can not be assumed to be certain but only exist with the associated probability.

The *attribute uncertainty* model on the other hand side assumes that each tuple is existent in the database, but the exact values of some attributes are not known for sure. Consider the previously mentioned witness example shown in Table 8.2. Here the witnesses are all sure that they have seen a car leaving an accident, just the brand (e.g. John saw a Mazda with 20% or a Toyota with 80% probability) and the color (Bob saw a red car with 70% or an orange with 30% probability) are not always known for sure. Following the *attribute uncertainty* model the probabilities for all possible values of an attribute of a tuple sum up to 1.0. When using *attribute uncertainty* to model the uncertainty, the models can further be classified in two types: *discrete* and *continuous* uncertainty models. *Discrete models* represent each uncertain attribute by a discrete set of alternative values, each associated with a probability. This model is in general adopted for probabilistic databases, where tuples are associated with existential probabilities , e.g.[57, 108, 146, 83]. In the *continuous* model an uncertain attribute is represented by a probability density function (*pdf*) over the value space. Figure 8.1 illustrates the difference between the two characteristics for one uncertain attribute.

It is also possible to combine *attribute* and *tuple* uncertainty which is shown by the example in Table 8.3. Here sensors s1, s2 and s3 capture values which are not certain, but rather a set of possible values assigned with a probability is given. The probabilities of the sensor readings of s2 do not sum up to 1 which implies existential uncertainty of the reading (i.e. it is possible that the sensor has not transmitted a sensor reading at all).

8.2.2 The Possible Worlds Semantics

Nowadays the most common way to treat this uncertainty in databases is the *possible worlds* semantics [99]. Many works [1, 140, 86] view a probabilistic database as a set of



(a) Discrete attribute uncertainty

(b) Continuous attribute uncertainty

Figure 8.1: Variants of attribute uncertainty

ID	**Value**
s1	$[3 : 0.3] \oplus [8 : 0.7]$
s2	$[6 : 0.5]$
s3	$[1 : 0.2] \oplus [2 : 0.1] \oplus [4 : 0.7]$

Table 8.3: Sensor readings

possible instances (worlds) associated with their probabilities. In the scope of this chapter (and also most existing works) we assume attribute values (which are random values) of different tuples to be mutually independent. However attribute values of the same tuple may not be independent from each other.

Definition 8.1 (Possible Worlds)**.** $\mathcal{D} = \{T_1, \dots T_N\}$ be an uncertain database and let $W = t_1, \dots, t_n$ be any (certain) database instance which corresponds to a subset of tuples t_i appearing in $\mathcal{D}$ such that $t_i \in T_i, i \in \{1, \dots, N\}$. The probability of this database instance (world) W to occur is $P(W) = \Pi_{i=1}^{N} P(t_i)$. If P(W) > 0, W is a possible world: the set of all possible worlds is denoted by $\mathcal{W}$.

Considering the example from Table 8.3 we can generate the set of possible worlds as shown in Table 8.4. Under possible worlds semantics it is now possible to answer queries and assign probabilities to the results. Again considering the example shown in Table 8.3, we may issue the following query: "Is the value of sensor s1 larger than the value of sensor s3?". For computing the answer we have to investigate all *possible worlds* and sum up the probabilities for each possible outcome. The result to this query is NO with a probability of 0.21 ($P(W_3) + P(W_6)$) and YES with a probability of 0.79 ($\Sigma_{W_i \in \mathcal{W} \setminus W_3, W_6} P(W_i)$).

8.2.3 Uncertain Spatial Data Model

Up to now we have considered uncertain relational data, but in this part we want to concentrate on uncertain spatial data. The thoughts and basics from above can however straightforwardly be adapted to the multidimensional spatial case. Following, we assume that the database $\mathcal{D}$ consists of multi-attribute objects $U_1, ..., U_N$ that may have uncertain attribute values. An uncertain attribute is defined as follows:

World	**Tuples**	**Prob.**
W_1	$\{s1 = 3\}, \{s2 = 6\}, \{s3 = 1\}$	0.03
W_2	$\{s1 = 3\}, \{s2 = 6\}, \{s3 = 2\}$	0.015
W_3	$\{s1 = 3\}, \{s2 = 6\}, \{s3 = 4\}$	0.105
W_4	$\{s1 = 3\}, \{s3 = 1\}$	0.03
W_5	$\{s1 = 3\}, \{s3 = 2\}$	0.015
W_6	$\{s1 = 3\}, \{s3 = 4\}$	0.105
W_7	$\{s1 = 8\}, \{s2 = 6\}, \{s3 = 1\}$	0.07
W_8	$\{s1 = 8\}, \{s2 = 6\}, \{s3 = 2\}$	0.035
W_9	$\{s1 = 8\}, \{s2 = 6\}, \{s3 = 4\}$	0.245
W_{10}	$\{s1 = 8\}, \{s3 = 1\}$	0.07
W_{11}	$\{s1 = 8\}, \{s3 = 2\}$	0.035
W_{12}	$\{s1 = 8\}, \{s3 = 4\}$	0.245

Table 8.4: Possible worlds

Definition 8.2 (Uncertain Attribute)**.** A uncertain attribute *attr* of object U_i is a random variable drawn from a probability distribution with density function f_i^{attr}.

Based on this definition we are able to define an uncertain object.

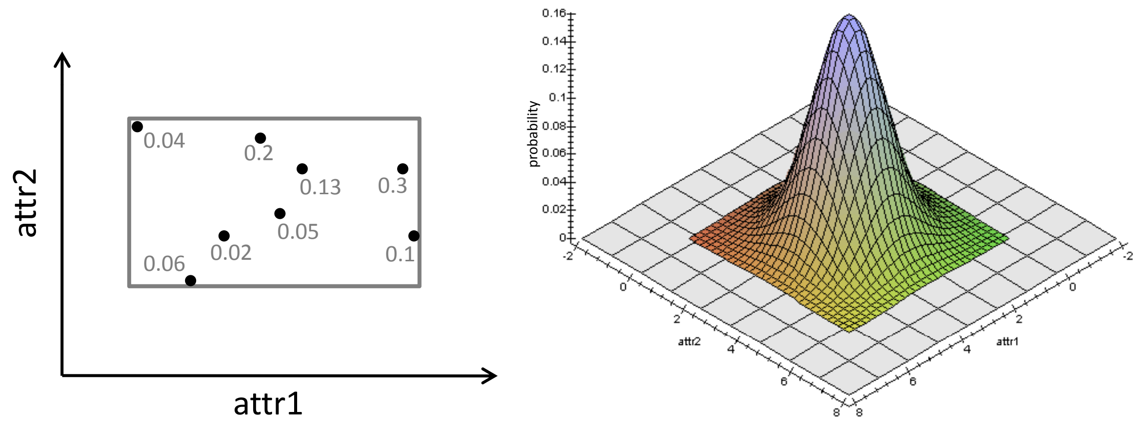
Definition 8.3 (Uncertain Object)**.** An uncertain object U_i has at least one uncertain attribute value. The function f_i denotes the multi-dimensional probability density distribution of U_i that combines all density functions for all probabilistic attributes *attr* of U_i. Thus U_i can be interpreted as a random variable.

In practice the spatial location of an object is assumed to be a random variable with a given probability distribution over all possible locations, either in terms of continuous distributions [155, 127, 32] or discrete distributions [96, 97]. Discrete distributions represent each uncertain object by a discrete set of alternative values (cf Figure 8.2(a)), each associated with a probability [112, 32]. In the continuous case each uncertain object is given by a multi-dimensional PDF (c.f. Figure 8.2(b), where the uncertain attributes may be mutually dependent. Therefore the object PDF can have any arbitrary form, and in general, cannot simply be derived from the marginal distribution of the uncertain attributes. Methods based on continuous distributions. Operations on this model usually involve expensive integrations, hence typically special approximation and indexing techniques for efficient query processing are employed [55, 149, 30]

Following the convention of uncertain databases [35, 46, 50, 52, 57, 110, 145], we assume that f_i is (minimally) bounded by an *uncertainty region* R^i such that $\forall x \notin R^i : f_i(x) = 0$ and

$$\int_{R^i} f_i(x)dx \leq 1.$$

Specifically, the case $\int_{R^i} f_i(x)dx < 1$ implements existential uncertainty, i.e. object U_i may not exist in the database at all with a probability greater than zero. In the following



(a) Discrete uncertain object (b) Continuous uncertaint object

Figure 8.2: Variants of uncertain objects

we focus on the case $\int_{R^i} f_i(x)dx = 1$, but the proposed concepts can be easily adapted to existentially uncertain objects. Although the proposed approaches are also applicable for unbounded PDF, e.g., Gaussian PDF, here we assume f_i exceeds zero only within a bounded region. This is a realistic assumption because the spectrum of possible values of attributes is usually bounded and it is commonly used in related work, e.g. [46, 50] and [35]. Even if f_i is given as an unbounded PDF, a common strategy is to truncate PDF tails with negligible probabilities and normalize the resulting PDF. In specific, [35] shows that for a reasonable low truncation threshold, the impact on the accuracy of probabilistic ranking queries is quite low while having a very high impact on the query performance.

8.3 Related Work

Uncertainty in databases is a relatively new field and has received a lot of attention in the past few years. The main challenges here are data representation [35, 50, 6, 128] and efficient query processing [108, 147]. While the main goal for most similarity queries on certain data is to minimize the I/O-cost, in the context of uncertain data, the CPU-cost also have a dramatic impact on the overall runtime. Probabilistic similarity query processing on uncertain relational databases has focused on various aspects, with top-k and ranking queries being the most important and well researched class of queries [107, 57, 108, 83, 146, 32].

Probabilistic queries on uncertain spatial databases retrieve, for a given query object, those objects from the database having a high probability of satisfying some spatial query predicate. In particular approaches for probabilistic distance range queries [149], probabilistic nearest-neighbour (kNN) queries [52, 97, 85], probabilistic reverse nearest- neighbour queries [45, 110] as well as probabilistic similarity join queries [96] have been proposed. To reduce computational effort, [50] add threshold constraints in order to retrieve only objects whose probability of being the nearest neighbor exceeds a user-specified threshold to control the desired confidence required in a query answer. In order to support probabilistic spatial queries efficiently, methods for indexing uncertain spatial data have been investigated. The most prominent index for uncertain spatial data is the U-Tree [149, 155] and its extension [179] which bound each spatial uncertain object with an MBR and additionally associates it with a set of "probabilistically constrained regions" (PCR). PCRs are rectangular regions serving as a tight conservative spatial approximation of the underlying uncertain object's spatial distribution. These PCRs can be used for probabilistic pruning during query processing. For efficiency reasons, the set of PCRs are conservatively approximated by a linear function over the parameters of the PCRs. Böhm et al. have introduced the Gauss-Tree [40], an index for locations whose distributions follow a Gaussian function. An approach for efficient probabilistic ranking based on this index is shown in [41]. Most of the existing approaches are however restricted to a certain query object. The few existing approaches allowing uncertainty of the query either use expected distances [114] or other methods which do not return results according to the *possible world semantics* and may thus produce very inaccurate results, that may have a very small probability of being an actual result ([146, 108]). Existing solutions on probabilistic k-nearest neighbor (kNN) queries restrict to expected distances of the uncertain objects to the query object [114] or also use a threshold constraint [51]. However, the use of expected distances does not adhere to the *possible world semantics* and may thus produce very inaccurate results, that may have a very small probability of being an actual result ([146, 108]). To the best of our knowledge, k-nearest neighbor queries, reverse k-nearest neighbour queries as well as ranking queries on uncertain data, where the query object is allowed to be uncertain and the *possible world semantics* is considered, have not been addressed so far. Since we have already observed that spatial domination is a practical tool for improving the runtime of many similarity queries, we want to investigate the concept of domination on uncertain data. In the following the goal is to boost all of the above mentioned similarity queries when using the domination criteria as a pruning criterion.

8.4 Distance Dependencies in Uncertain Spatial Data

Uncertain objects as defined in Definition 8.3 are obviously subject to the same distance dependencies than extended spatial objects (actually they are a special case of extended spatial objects) during similarity query processing. Uncertain objects however provide much more information than simple spatial approximations, thus it is possible to refine the concept of domination for this special data type.

8.4.1 Probabilistic Domination

Again we are interested in the following variation of the domination problem: Given three uncertain objects A, B and R in a multidimensional space $\mathbb{R}^d$, determine whether object A is closer to R than B w.r.t. a distance function defined on the objects in $\mathbb{R}^d$. If this is the case, we say A *dominates* B w.r.t. R. In contrast to Part II, where this problem

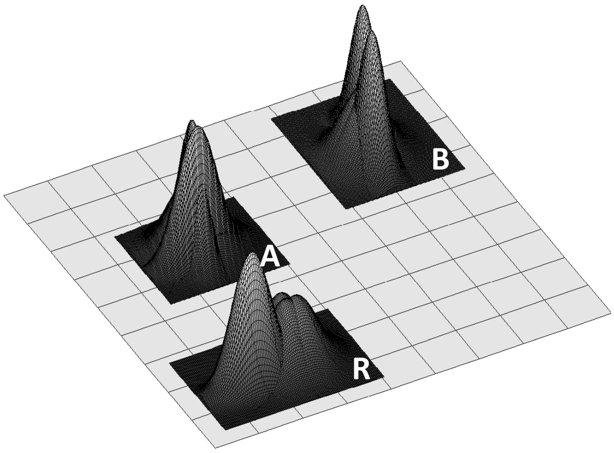


Figure 8.3: A dominates B w.r.t. R with high probability.

is solved for certain data, in the context of uncertain objects this domination relation is not a predicate that is either true or false, but rather a (dichotomous) random variable as defined in the following definition.

Definition 8.4 (Domination Random Indicator Variable)**.** Let A, B and R be uncertain objects. $A \prec_R B$ is the random indicator variable that is 1, iff A dominates B w.r.t. R, formally:

$$A \prec_R B = \begin{cases} 1, & \text{if } a \in A, b \in B, r \in R : \mathit{dist}(a,r) < \mathit{dist}(b,r) \\ 0, & \text{otherwise} \end{cases}$$

where a, b and r are realizations of the random variables A, B and R, respectively and dist is a distance function on vector objects.

In the example depicted in Figure 8.3, there are three uncertain objects A, B and R, each bounded by a rectangle representing the possible locations of the object in $\mathbb{R}^2$. The PDFs of A, B and R are depicted as well. In this scenario, we cannot determine for sure whether object A dominates B w.r.t. R. However, it is possible to determine that object A dominates object B w.r.t. R with a high probability. The problem at issue is to determine the *probabilistic domination* relation defined as:

Definition 8.5 (Probabilistic Domination)**.** Given three uncertain objects A, B and R, the probabilistic domination $P(A \prec_R B = 1)$ denotes the probability that A dominates B w.r.t. R. For simplicity this probability will be denoted as $P(A \prec_R B)$.

Naively, we can compute $P(A \prec_R B)$ by "simply" integrating the probability of all possible worlds in which A dominates B w.r.t. R exploiting inter-object independency:

$$P(A \prec_R B) = \int_{a \in A} \int_{b \in B} \int_{r \in R} \delta(a,b,r) \cdot P(A = a) \cdot P(B = b) \cdot P(R = r) da\, db\, dr,$$

where $\delta(a, b, r)$ is the following indicator function:

$$\delta(a, b, r) = \begin{cases} 1, & \text{if } dist(a, r) < dist(b, r) \\ 0, & \text{else} \end{cases}$$

The problem of this naive approach is the computational cost of the three-fold-integral. The integrals of the PDFs of A, B and R may in general not be representable as a closed-form expression and the integral of $A \prec_R B$ does not have a closed-from expression. Therefore, an expensive numeric approximation is required for this approach.

Obviously it is easy to show how to detect whether A completely dominates B w.r.t. R (i.e. if $P(A \prec_R B) = 1$) regardless of the probability distributions assigned to the rectangular uncertainty regions. Many existing works still use DDC$_{MM}$ to detect this relation [122, 97]. Although correct, this criterion is not complete (see Part II of this work) in all cases, i.e. not each case where A dominates B w.r.t. R is detected by the criterion. Therefore, we adopt the spatial domination concepts proposed in Chapter 7 (Definition 7.1) for rectangular uncertainty regions.

Corollary 8.1 (Complete Probabilistic Domination)**.** Let A, B, R be uncertain objects with rectangular uncertainty regions. Then the following statement holds:

$$P(A \prec_R B) = 1 \Leftrightarrow \sum_{i=1}^{d} \max_{r_i \in \{R_i^{min}, R_i^{max}\}} (MaxDist(A_i, r_i)^p - MinDist(B_i, r_i)^p) < 0,$$

where A_i, B_i and R_i denote the projection interval of the respective rectangular uncertainty region of A, B and R on the i^{th} dimension; R_i^{min} (R_i^{max}) denotes the lower (upper) bound of interval R_i, and p corresponds to the used L_p norm. The functions $MaxDist(A, r)$ and $MinDist(A, r)$ denote the maximal (respectively minimal) distance between the one-dimensional interval A and the one-dimensional point r.

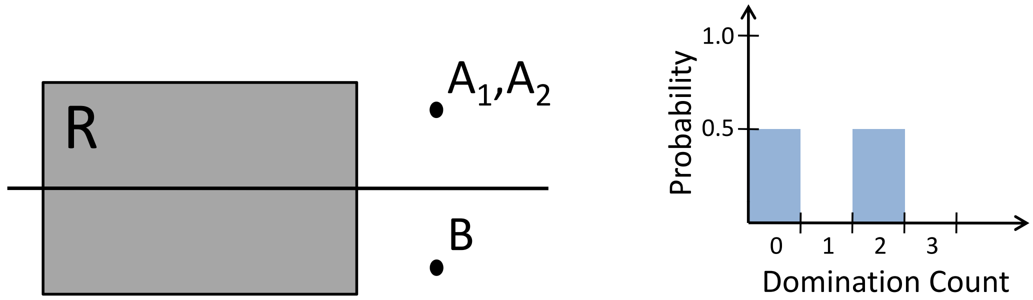
Corollary 8.1 follows directly from the considerations of the optimal spatial domination decision criterion (DDC$_{OPT}$) in Chapter 7; the inequality is true if and only if for all points $a \in A, b \in B, r \in R$, a is closer to r than b. Translated into the possible worlds model, this is equivalent to the statement that A is closer to R than B for any possible world, which in return means that $P(A \prec_R B) = 1$.

In addition, it holds that

Corollary 8.2.

$$P(A \prec_R B) = 1 \Leftrightarrow P(B \prec_R A) = 0$$

The complete probabilistic domination is only a special case of the probabilistic domination relation. Determining $P(A \prec_R B)$ in the general case is a complex problem involving costly integration. In Chapters 9 and 10 we will therefore present techniques for approximating this probability in order to further use the outcome to compute the probabilistic domination count.



(a) A_1 and A_2 dominate B w.r.t. R with a probability of 50%, respectively.

(b) The corresponding probabilistic domination count

Figure 8.4: Example of probabilistic domination

8.4.2 Probabilistic Domination Count on Uncertain Object Databases

After being able to correctly compute probabilistic domination the next challenge at hand is how to derive the probabilistic domination count.

Definition 8.6 (Probabilistic Domination Count)**.** Consider an uncertain database $\mathcal{D} = \{o_1, ..., o_N\}$ and an uncertain reference object R. For each uncertain object $B \in \mathcal{D}$, let $PDC(\mathcal{D}, B, R)$ be the random variable of the number of uncertain objects $A \in \mathcal{D}$ ($A \neq B$) that are closer to R than B:

$$PDC(\mathcal{D}, B, R) = \sum_{A \in \mathcal{D}, A \neq B} A \prec_R B$$

$PDC(\mathcal{D}, B, R)$ is the sum of $N - 1$ non-necessarily identically distributed and non-necessarily independent Bernoulli variables. The problem to be solved is how to efficiently compute the probability density distribution of $PDC(\mathcal{D}, B, R)$.

As already mentioned determining the domination count is a central module for most types of similarity queries in order to identify true hits and true drops (pruning). In the context of probabilistic similarity queries, knowledge about the PDF of $PDC(\mathcal{D}, B, R)$ can be used to find out if B satisfies the query predicate. Computing $PDC(\mathcal{D}, B, R)$ however is not straightforwardly possible even when we are able to compute $P(A \prec_R B)$ for three objects A, B and R. To give an intuition how challenging the problem is, we following present a naive solution that can yield incorrect results due to ignoring dependencies between domination relations.

Example 8.4.1. Consider a database of three certain objects B, A_1 and A_2 and the uncertain reference object R, as shown in Figure 8.4(a). For simplicity, objects A_1 and A_2 have the same position in this example. The task is to determine the domination count of B w.r.t. R. The domination half-space ($H_{A_{1/2}B}(A_{1/2})$) for A_1 and A_2 is depicted here as well (the upper halfspace). Let us assume that A_1 (A_2) dominates B with a probability of $P(A_1 \prec_R B) = P(A_2 \prec_R B) = 50\%$. Recall that this probability can be computed by integration. However, in this example, the probability that both A_1 and A_2 dominate B is not simply $P(A_1 \prec_R B) \cdot P(A_2 \prec_R B) = 50\% \cdot 50\% = 25\%$.

The reason for the wrong result in this example is that this kind of computation requires mutually independent random variables. However, in this example, it holds that if and only if R falls into the domination half-space $H_{A_1B}(A_1)$, it also falls into the domination half-space $H_{A_2B}(A_2)$. Thus we have the dependency $A_1 \prec_R B \leftrightarrow A_2 \prec_R B$ and the probability for R to be dominated by both A_1 and A_2 is according to the multiplication theorem of probability

$$P(A_1 \prec_R B \cap A_2 \prec_R B) = P(A_1 \prec_R B) \cdot P(A_2 \prec_R B)|A_1 \prec_R B) = 0.5 \cdot 1 = 0.5.$$

The above simple approach thus ignores possible dependencies between domination relationships. Although we assume independence between objects, the random variables $A_1 \prec_R B$ and $A_2 \prec_R B$ are mutually dependent because the distance between A_1 and R depends on the distance between A_2 and R because object R can only appear once. The correct probabilistic domination count of B w.r.t. R for the example is is therefore either 0 or 2 depending on the position of R (cf. Figure 8.4(b)).

8.5 Challenges of Distance Dependencies in Uncertain Data

The rest of this part is separated into two chapters. In Chapter 9 we will build the theoretical foundation for computing probabilistic domination and the probabilistic domination count. Subsequently we will show in Chapter 10 how the proposed computation can be plugged into the application of probabilistic reverse nearest neighbour processing on discrete uncertain data. In particular this part will focus on the following challenges:

As mentioned in Section 8.4.1 the naive computation of $P(A \prec_R B)$ is very costly in the general case since it involves a three-fold-integration over the pdfs of the involved objects. Thus the first step is to find an efficient way to approximate this value for three given uncertain objects A, B and R. In Example 8.4.1 we showed that it is not possible to use the values $P(A_i \prec_R B)$ naively in order to compute the probabilistic domination count $PDC(\mathcal{D}, B, R)$ for objects $A_i \in \mathcal{D}, B$ and R. However we will show that it is possible to obtain the approximations of all $P(A_i \prec_R B)$ in a way that they can be offset with each other. This operation however is not trivial and we will develop a technique which we term *uncertain generating functions* (UGFs). These UGFs allow for an efficient approximation of the probabilistic domination count, which is sufficient in most applications.

As a proof of efficiency for the proposed techniques we chose the application of probabilistic reverse nearest neighbour processing on discrete uncertain data. For this purpose we develop a framework which allows for efficient computations of a broad class of probabilistic similarity queries. We adapt the developed techniques from Chapter 9 to the special case of the discrete uncertainty model and show how the probabilistic domination count can be utilized as pruning criterion to develop algorithms which are superior to existing approaches.

Chapter 9

Probabilistic Domination on Continuous Uncertain Data

In this chapter, we show how to efficiently approximate (i.e. upper and lower bound) the probabilistic domination $P(A \prec_R B)$ of three objects A, B and R. Furthermore we will show how to utilize these bounds in order to compute bounds of the *probabilistic domination count* $PDC(\mathcal{D}, B, R)$. The approach supports a general uncertainty model using continuous probabilistic density functions to describe the (possibly correlated) uncertain attributes of objects. Specifically, we propose an iterative filter-refinement strategy for conservatively and progressively estimating the probabilistic domination count in an efficient way while keeping correctness according to the possible worlds semantics. In an experimental evaluation, we show that our proposed technique allows to acquire tight probability bounds for the probabilistic domination count quickly, even for large uncertain databases.

Parts of this chapter have been published in [30]. The rest of this chapter is organized as follows: In Sections 9.1 and 9.2 we show how to compute bounds for *probabilistic domination* and the *probabilistic domination count*, respectively. In order to set the results off against each other we develop the technique of *generating functions* in Section 9.3 and show how it can be implemented efficiently in Section 9.4. Section 9.5 shows how the proposed methods can be integrated in different query predicates. An experimental evaluation is given in Section 9.6 before Section 9.7 concludes this chapter.

9.1 Bounding Probabilistic Domination

We start by considering the case where A does not completely dominate B w.r.t. R. In consideration of the possible world semantics, there may exist worlds in which A dominates B w.r.t. R, but not all possible worlds satisfy this criterion (corresponding to the concept of partial domination introduced in Definition 3.6). Let us consider the example shown in Figure 9.1 where the uncertainty region of A is decomposed into five partitions, each assigned to one of the five grey-shaded regions illustrating which points are closer to the partition in A than to B. As we can see, R only completely falls into three grey-shaded

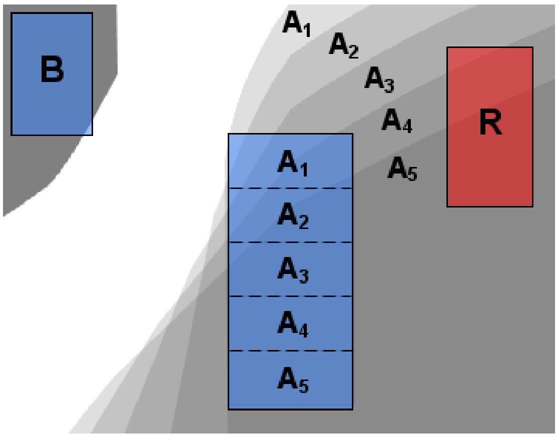


Figure 9.1: Probabilistic domination

regions. This means that A does not completely dominate B w.r.t. R. However, we know that in some possible worlds (at least in all possible worlds where A is located in A_1, A_2 or A_3) A does dominate B w.r.t. R. The question at issue is how to determine the probability $P(A \prec_R B)$ that A dominates B w.r.t. R in an efficient way. The key idea is to decompose the uncertainty region of an object X into subregions for which we know the probability that X is located in that subregion (as done for object A in our example). Therefore, if neither $Dom(A, B, R)$ nor $Dom(B, A, R)$ holds, then there may still exist subregions $A' \subset A$, $B' \subset B$ and $R' \subset R$ such that A' dominates B' w.r.t. R'. Given disjunctive decomposition schemes $\underline{\mathcal{A}}$, $\underline{\mathcal{B}}$ and $\underline{\mathcal{R}}$ we can identify triples of subregions ($A' \in \underline{\mathcal{A}}$, $B' \in \underline{\mathcal{B}}$, $R' \in \underline{\mathcal{R}}$) for which $Dom(A', B', R')$ holds. Let $\delta(A', B', R')$ be the following indicator function:

$$\delta(A', B', R') = \begin{cases} 1, & \text{if } Dom(A', B', R') \\ 0, & \text{else} \end{cases}$$

Lemma 9.1. Let A, B and R be uncertain objects with disjunctive object decompositions $\underline{\mathcal{A}}, \underline{\mathcal{B}}$ and $\underline{\mathcal{R}}$, respectively. To derive a lower bound $P_{LB}(A \prec_R B)$ of the probability $P(A \prec_R B)$ that A dominates B w.r.t. R, we can accumulate the probabilities of combinations of these subregions as follows:

$$P_{LB}(A \prec_R B) = \sum_{A' \in \underline{\mathcal{A}}, B' \in \underline{\mathcal{B}}, R' \in \underline{\mathcal{R}}} P(A') \cdot P(B') \cdot P(R') \cdot \delta(A', B', R'),$$

where $P(X')$ denotes the probability that object X is located within the partition X'.

Proof. The probability of a combination (A', B', R') can be computed by $P(A') \cdot P(B') \cdot P(R')$ due to the assumption of mutually independent objects. These probabilities can be aggregated due to the assumption of disjunctive subregions, which implies that any two different combinations of subregions $(A' \in \underline{\mathcal{A}}, B' \in \underline{\mathcal{B}}, R' \in \underline{\mathcal{R}})$ and $(A'' \in \underline{\mathcal{A}}, B'' \in \underline{\mathcal{B}}, R'' \in \underline{\mathcal{R}}$, $A' \neq A'' \vee B' \neq B'' \vee R' \neq R''$ must represent disjunctive sets of possible worlds. It is obvious that all possible worlds defined by combinations (A', B', R') where $\delta(A', B', R') = 1$, A dominates B w.r.t. R. But not all possible worlds where A dominates B w.r.t. R are covered by these combinations and, thus, do not contribute to $P_{LB}(A \prec_R B)$. Consequently, $P_{LB}(A \prec_R B)$ lower bounds $P(A \prec_R B)$. $\square$

Analogously, we can define an upper bound of $P(A \prec_R B)$:

Lemma 9.2. An upper bound $P_{UB}(A \prec_R B)$ of $P(A \prec_R B)$ can be derived as follows:

$$P_{UB}(A \prec_R B) = 1 - P_{LB}(B \prec_R A)$$

Proof. This equation is evident due to Lemma 9.1 and the observation that in any world where $A = a$, $B = b$ and $R = r$ it holds that $Dom(a, b, r) \Leftrightarrow \neg Dom(b, a, r)$. $\square$

Naturally, the more refined the decompositions are, the tighter the bounds that can be computed and the higher the corresponding cost of deriving them. In particular, starting from the entire MBRs of the objects, we can progressively partition them to iteratively derive tighter bounds for their dependency relationships until a desired degree of certainty is achieved (based on some threshold). However using the retrieved bounds directly for computing the probabilistic domination count still yields the same problems as already discussed in the Section 8.4.2 (where we stated that the exact values $P(A_i \prec_R B)$ can not be used to calculate the probabilistic domination count $PDC(\mathcal{D}, B, R)$). In the next section, we show how to derive bounds for the probabilistic domination count $PDC(\mathcal{D}, B, R)$ of a given object B (cf. Definition 8.6) by using a methodology based on *generating functions*.

9.2 Bounding the Probabilistic Domination Count

In general, domination relations may have arbitrary correlations. Therefore, we present a way to compute the probabilistic domination count $PDC(\mathcal{D}, B, R)$ while accounting for the dependencies between domination relations. The basic idea here is to use bounds for the probabilistic domination which are mutually independent from each other.

9.2.1 Complete Domination

In an initial step, *complete domination* (as mentioned in Corollary 8.1) serves as a filter which allows us to detect those objects $A \in \mathcal{D}$ that definitely dominate a specific object B w.r.t. R and those objects that definitely do not dominate B w.r.t. R by means of evaluating $P(A \prec_R B)$ and $P(B \prec_R R)$ respectively. We are able to achieve this for example by using DDC$_{OPT}$from Part II of this work. It is important to note that complete domination relations are mutually independent, since complete domination is evaluated on the entire uncertainty regions of the objects. After applying complete domination, we have detected objects that dominate B in all, or no possible worlds. Consequently, we get a first approximation of the probabilistic domination count $PDC(\mathcal{D}, B, R)$, obviously, it must be higher than the number N of objects that completely dominate B and lower than $|\mathcal{D}| - M$, where M is the number of objects that dominate B in no possible world, i.e. $P(PDC(\mathcal{D}, B, R) = k) = 0$ for $k \leq N$ and $k \geq |\mathcal{D}| - M$. Nevertheless, for $N < k < |\mathcal{D} - M|$ we still have a very bad approximation of the domination count probability of $0 \leq P(PDC(\mathcal{D}, B, R) = k) \leq 1$.

9.2.2 Probabilistic Domination

In order to refine this probability distribution, we have to take the remaining set of objects $influenceObjects = \{A_1, ..., A_C\}$ influencing the probabilistic domination count. In particular the $influenceObjects$ set contains objects which neither completely dominate B nor are completely dominated by B w.r.t. R. Consequently for each $A_i \in influenceObjects$ the probability $P(A_i \prec_R B)$ may be unequal 0 or 1. For these objects, we can compute probabilities $P(A_1 \prec_R B), ..., P(A_C \prec_R B)$ by integration or bound the probabilities according to the methodology in Section 9.1. However, due to the mutual dependencies between domination relations (cf. Section 8.4.2), we cannot simply use these probabilities directly, as they may produce incorrect results. However, we can use the observation that the objects A_i are mutually independent and each candidate object A_i only appears in a single domination relation $A_1 \prec_R B, ..., A_C \prec_R B$. Exploiting this observation, we can decompose the objects $A_1, ..., A_C$ only, to obtain mutually independent bounds for the probabilities $P(A_1 \prec_R B), ..., P(A_C \prec_R B)$, as stated by the following lemma:

Lemma 9.3. Let $A_1, ...A_C$ be uncertain objects with disjunctive object decompositions $\mathcal{A}_1, ..., \mathcal{A}_C$, respectively. Also, let B and R be uncertain objects (without any decomposition). The lower (upper) bound $P_{LB}(A_i \prec_R B)$ ($P_{UB}(A_i \prec_R B)$) as defined in Lemma 9.1 (Lemma 9.2) of the random variable $A_i \prec_R B$ is independent of the random variable $A_j \prec_R B$ $(1 \leq i \neq j \leq C)$.

Proof. Consider the random variable $A_i \prec_R B$ conditioned on the event $A_j \prec_R B = 1$. Using Equation 9.1, we can derive the lower bound probability of $A_i \prec_R B = 1$ conditioned to the event $A_j \prec_R B = 1$ as follows:

$$P_{LB}(A_i \prec_R B | A_j \prec_R B) =$$
$$\sum_{A'_i \in \mathcal{A}_i, B' \in \mathcal{B}, R' \in \mathcal{R}} [P(a_i \in A'_i | A_j \prec_R B) \cdot P(b \in B' | A_j \prec_R B) \cdot P(r \in R' | A_j \prec_R B) \cdot \delta(A'_i, B', R')]$$

Now we exploit that B and R are not decomposed, thus $B' = B$ and $R' = R$, and thus $P(B \in B' | A_j \prec_R B) = 1 = P(B \in B')$ and $P(R \in R' | A_j \prec_R B) = 1 = P(R \in R')$. We obtain:

$$P_{LB}(A_i \prec_R B | A_j \prec_R B = 1) =$$
$$\sum_{A'_i \in \mathcal{A}_i, B' \in \mathcal{B}, R' \in \mathcal{R}} [P(a_i \in A'_i | A_j \prec_R B = 1) \cdot P(b \in B') \cdot P(r \in R') \cdot \delta(A'_i, B', R')]$$

Next we exploit that $P(a_i \in A'_i | A_j \prec_R B) = P(a_i \in A'_i)$ since A_i is independent from $A_j \prec_R B$ and obtain:

$$P_{LB}(A_i \prec_R B | A_j \prec_R B) = \sum_{A'_i \in \mathcal{A}_i, B' \in \mathcal{B}, R' \in \mathcal{R}} [P(a_i \in A'_i) \cdot P(b \in B') \cdot P(r \in R') \cdot \delta(A'_i, B', R')] = P_{LB}(A_i \prec_R B)$$

Analogously, it can be shown that

$$P_{UB}(A_i \prec_R B | A_j \prec_R B = 1) = P_{UB}(A_i \prec_R B).$$

□

In summary, we can now derive, for each object A_i a lower and an upper bound of the probability that A_i dominates B w.r.t. R. However, these bounds may still be rather loose, since we only consider the full uncertainty region of B and R so far, without any decomposition. In Section 9.3.5, we will show how to obtain more accurate, still mutual independent probability bounds based on decompositions of B and R. Due to the mutual independency of the lower and upper probability bounds, these probabilities can then be used to get an approximation of the domination count of B. In order to do this efficiently, we adapt the generating functions technique which is proposed in [108]. The main challenge here is to extend the generating function technique in order to cope with probability bounds instead of concrete probability values. It can be shown that a straightforward solution based on the existing generating functions technique applied to the lower/upper probability bounds in an appropriate way does solve the given problem efficiently, but overestimates the domination count probability and thus, does not yield good probability bounds (We will reveal the problems of this approach in Section 9.3.4). Rather, we have to redesign the generating functions technique such that lower/upper probability bounds can be handled correctly.

9.3 Uncertain Generating Functions (UGFs)

In this section, we will give a brief survey on the existing generating function technique (for more details refer to [108]) and then propose our new technique of uncertain generating functions.

9.3.1 Generating Functions

Consider a set of N *mutually independent*, but not necessarily identically distributed Bernoulli $\{0, 1\}$ random variables $X_1, ..., X_N$. Let $P(X_i)$ denote the probability that $X_i = 1$. The problem is to efficiently compute the sum

$$\sum_{i=1}^{N} X_i = \sum_{i=1}^{N} A_i \prec_R B$$

of these random variables. A naive solution would be to count, for each $0 \leq k \leq N$, all combinations with exactly k occurrences of $X_i = 1$ and accumulate the respective probabilities of these combinations. This approach, however, shows a complexity of $O(2^N)$. In [34], an approach was proposed that achieves an $O(N)$ complexity using the *Poisson Binomial Recurrence.* Note that $O(N)$ time is asymptotically optimal in general, since the computation involves at least $O(N)$ computations, namely $P(X_i), 1 \leq i \leq N$. In the following, we propose a different approach that, albeit having the same linear asymptotical complexity, has other advantages, as we will see. We apply the concept of generating functions as proposed in the context of probabilistic ranking in [108]. Consider the function $\mathcal{F}(x) = \prod_{i=1}^{n}(a_i+b_ix)$. The coefficient of x^k in $\mathcal{F}(x)$ is given by: $\sum_{|\beta|=k} \prod_{i:\beta_i=0} a_i \prod_{i:\beta_i=1} b_i$, where $\beta = \langle \beta_1, ..., \beta_N \rangle$ is a Boolean vector, and $|\beta|$ denotes the number of 1's in β.

Now consider the following generating function:

$$\mathcal{F}^i = \prod_{X_i}(1 - P(X_i) + P(X_i) \cdot x) = \sum_{j \geq 0} c_j x^j.$$

The coefficient c_j of x^j in the expansion of $\mathcal{F}^i$ is the probability that for exactly j random variables X_i it holds that $X_i = 1$. Since $\mathcal{F}^i$ contains at most $i+1$ non-zero terms and by observing that

$$\mathcal{F}^i = \mathcal{F}^{i-1} \cdot (1 - P(X_i) + P(X_i) \cdot x),$$

we note that $\mathcal{F}^i$ can be computed in $O(i)$ time given $\mathcal{F}^{i-1}$. Since $\mathcal{F}^0 = 1x^0 = 1$, we conclude that $\mathcal{F}^N$ can be computed in $O(N^2)$ time. If only the first k coefficients are required (i.e. coefficients c_j where $j < k$), this cost can be reduced to $O(k \cdot N)$, by simply dropping the terms of the sum $c_j x^j$ where $j \geq k$.

Example 9.3.1. As an example, consider three *independent* random variables X_1, X_2 and X_3. Let X_i correspond to the event that $A_i \prec_R B$. Furthermore for the sake of simplicity we assume that the random variables X_i are mutually independent (which is not the case in general and will be addressed later). Let $P(X_1) = 0.2$, $P(X_2) = 0.1$ and $P(X_3) = 0.3$, and let $k = 2$. Then:

$$\mathcal{F}^1 = \mathcal{F}^0 \cdot (0.8 + 0.2x) = 0.2x^1 + 0.8x^0$$

$$\mathcal{F}^2 = \mathcal{F}^1 \cdot (0.9 + 0.1x) = 0.02x^2 + 0.26x^1 + 0.72x^0 \overset{*}{=} 0.26x^1 + 0.72x^0$$

$$\mathcal{F}^3 = \mathcal{F}^2 \cdot (0.7 + 0.3x) = 0.078x^2 + 0.418x^1 + 0.504x^0 \overset{*}{=} 0.418x^1 + 0.504x^0$$

Thus, $P(PDC(\mathcal{D}, B, R) = 0) = 50.4\%$ and $P(PDC(\mathcal{D}, B, R) = 1) = 41.8\%$. We obtain $P(PDC(\mathcal{D}, B, R) < 2) = 92.2\%$. Equations marked by * exploit that we only need to compute the c_j where $j < k = 2$.

9.3.2 Uncertain Generating Functions

Given a set of N independent but not necessarily identically distributed Bernoulli $\{0, 1\}$ random variables $X_i, 1 \leq i \leq N$. Let $P_{LB}(X_i)$ ($P_{UB}(X_i)$) be a lower (upper) bound approximation of the probability $P(X_i = 1)$. Consider the random variable

$$\sum_{i=1}^{N} X_i.$$

We make the following observation: The lower and upper bound probabilities $P_{LB}(X_i)$ and $P_{UB}(X_i)$ correspond to the probabilities of the three following events:

- $X_i = 1$ definitely holds with a probability of at least $P_{LB}(A_i <_R B)$.
- $X_i = 0$ definitely holds with a probability of at least $1 - P_{UB}(X_i)$.
- It is unknown whether $X_i = 0$ or $X_i = 1$ with the remaining probability of $P_{UB}(A_i <_R B) - P_{LB}(A_i <_R B)$.

Based on this observation, we consider the following uncertain generating function (UGF):

$$\mathcal{F}^N = \prod_{i \in 1,\ldots,N} [(P_{LB}(X_i) \cdot x + (1 - P_{UB}(X_i)) \cdot y + (P_{UB}(X_i) - P_{LB}(X_i)))] = \sum_{i,j \geq 0} c_{i,j} x^i y^j.$$

The coefficient $c_{i,j}$ has the following meaning: With a probability of $c_{i,j}$, B is definitely dominated at least i times, and possibly dominated another 0 to j times. Therefore, the minimum probability that $\sum_{i=1}^{N} X_i = k$ is $c_{k,0}$, since that is the probability that exactly k random variables X_i are 1. The maximum probability that $\sum_{i=1}^{N} X_i = k$ is $\sum_{i \leq k, i+j \geq k} c_{i,j}$, i.e. the total probability of all possible combinations in which $\sum_{i=1}^{N} X_i = k$, may hold. Therefore, we obtain an approximate PDF of $\sum_{i=1}^{N} X_i$. In the approximated PDF of $\sum_{i=1}^{N} X_i$, each probability $\sum_{i=1}^{N} X_i = k$ is given by a conservative and a progressive approximation.

Example 9.3.2. Let $P_{LB}(X_1) = 20\%$, $P_{UB}(X_1) = 50\%$, $P_{LB}(X_2) = 60\%$ and $P_{UB}(X_2) = 80\%$. The generating function for the random variable $\sum_{i=1}^{2} X_i$ is the following:

$$\mathcal{F}^2 = (0.2x + 0.5y + 0.3)(0.6x + 0.2y + 0.2) = 0.12x^2 + 0.34x + 0.1 + 0.22xy + 0.16y + 0.06y^2$$

That implies that, with a probability of at least 12%, $\sum_{i=1}^{2} X_i = 2$. In addition, with a probability of 22% plus 6%, it may hold that $\sum_{i=1}^{2} X_i = 2$, so that we obtain a probability bound of $12\% - 40\%$ for the random event $\sum_{i=1}^{2} X_i = 2$. Analogously, $\sum_{i=1}^{2} X_i = 1$ with a probability of $34\% - 78\%$ and $\sum_{i=1}^{2} X_i = 0$ with a probability of $10\% - 32\%$. The approximated PDF of $\sum_{i=1}^{2} X_i$ is depicted in Figure 9.2.

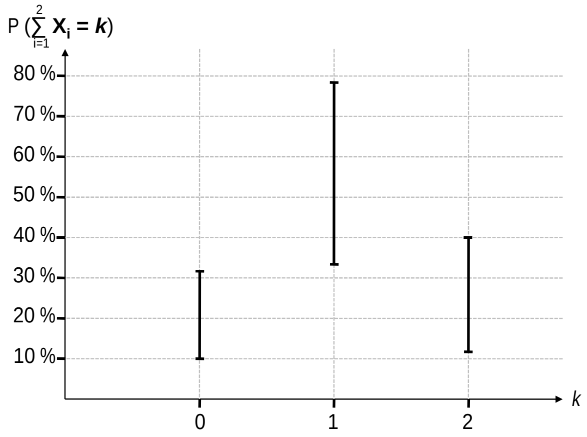


Figure 9.2: Approximated PDF of $\sum_{i=1}^{2} X_i$.

Each expansion $\mathcal{F}^l$ can be obtained from the expansion of $\mathcal{F}^{l-1}$ as follows:

$$\mathcal{F}^l = \mathcal{F}^{l-1} \cdot [P_{LB}(X_l) \cdot x + (1 - P_{UB}(X_l)) + (P_{UB}(X_l) - P_{LB}(X_l)) \cdot y].$$

We note that $\mathcal{F}^l$ contains at most $\sum_{i=1}^{l+1} i$ non-zero terms (one $c_{i,j}$ for each combination of i and j where $i + j \leq l$). Therefore, the total complexity to compute $\mathcal{F}^l$ is $O(l^3)$.

9.3.3 Efficient Domination Count Approximation using UGFs

We can directly use the uncertain generating functions proposed in the previous section to derive bounds for the probability distribution of the domination count $PDC(\mathcal{D}, B, R)$. Again, let $\mathcal{D} = A_1, ..., A_N$ be an uncertain object database and B and R be uncertain objects in $\mathbb{R}^d$. Let $A_i \prec_R B, 1 \leq i \leq N$ denote the random Bernoulli event that A_i dominates B w.r.t. R.[1] Also recall that the domination count is defined as the random variable that is the sum of the domination indicator variables of all uncertain objects in the database (cf. Definition 8.6).

Considering the generating function

$$\mathcal{F}^N = \prod_{i \in 1,...,N} [(P_{LB}(A_i \prec_R B) \cdot x + (P_{UB}(A_i \prec_R B) - P_{LB}(A_i \prec_R B)) \cdot y) + (1 - P_{UB}(A_i \prec_R B))]$$

$$= \sum_{i,j \geq 0} c_{i,j} x^i y^j \tag{9.1}$$

we can efficiently compute lower and upper bounds of the probability that $PDC(\mathcal{D}, B, R) = k$ for $0 \leq k \leq |\mathcal{D}|$, as discussed in Section 9.3.1 and because the independence property of random variables required by the generating functions is satisfied due to Lemma 9.3.

---

[1] That is, $X[A_i \prec_R B] = 1$ iff A_i dominates B w.r.t. R and $X[A_i \prec_R B] = 0$ otherwise.

Lemma 9.4. A lower bound $PDC^k_{LB}(\mathcal{D}, B, R)$ of the probability that $PDC(\mathcal{D}, B, R) = k$ is given by

$$PDC^k_{LB}(\mathcal{D}, B, R) = c_{k,0}$$

and an upper bound $PDC^k_{UB}(\mathcal{D}, B, R)$ of the probability that $PDC(\mathcal{D}, B, R) = k$ is given by

$$PDC^k_{LB}(\mathcal{D}, B, R) = \sum_{i \leq k, i+j \geq k} c_{i,j}$$

Example 9.3.3. Assume a database containing uncertain objects A_1, A_2, B and R. The task is to determine a lower (upper) bound of the probabilistic domination count probability $PDC^k_{LB}(\mathcal{D}, B, R)$ ($PDC^k_{UB}(\mathcal{D}, B, R)$) of B w.r.t. R. Assume that, by decomposing A_1 and A_2 and using the probabilistic domination approximation technique proposed in Section 9.1, we determine that A_1 has a minimum probability $P_{LB}(A_1 \prec_R B)$ of dominating B of 20% and a maximum probability $P_{UB}(A_1 \prec_R B)$ of 50%. For A_2, $P_{LB}(A_2 \prec_R B)$ is 60% and $P_{UB}(A_2 \prec_R B)$ is 80%. By applying the technique in the previous subsection, we get the same generating function as in Example 9.3.2 and thus, the same approximated PDF for the $PDC(\mathcal{D}, B, R)$ depicted in Figure 9.2.

To compute the uncertain generating function and thus the probabilistic domination count of an object in an uncertain database of size N, the total complexity is $O(N^3)$. The reason is that the maximum number of coefficients of the generating function $\mathcal{F}^x$ is quadratic in x, since $\mathcal{F}^x$ contains coefficients $c_{i,j}$ where $i + j \leq x$, that is at most $\frac{x^2}{2}$ coefficients. Since we have to compute $\mathcal{F}^x$ for each ($x < N$), the total time complexity is $O(N^3)$. Note that only objects $influenceObjects \subseteq \mathcal{D}$ for which a complete domination cannot be detected (cf. Section 9.2.1) have to be considered in the generating functions. Thus, the total runtime to compute $PDC^k_{LB}(\mathcal{D}, B, R)$ as well as $PDC^k_{UB}(\mathcal{D}, B, R)$ is $O(|influenceObjects|^3)$. In addition, we will show in Section 9.5 how to reduce the total time complexity to $O(k^2 \cdot |influenceObjects|)$ for certain similarity queries.

9.3.4 Generating Functions vs Uncertain Generating Functions

It is clear that instead of applying the uncertain generating function to approximate the domination count of B, two regular generating functions can be used; one generating function that uses the progressive (lower) bounds $P_{LB}(A_i \prec_R B)$ and one that uses the conservative (upper) probability bounds $P_{UB}(A_i \prec_R B)$. In the following we give an intuition and a formal proof that using regular generating functions yields looser bounds for the approximated probabilistic domination count.

Let $\mathcal{D} = A_1, ..., A_N$ be an uncertain object database and B and R be uncertain objects in $\mathbb{R}^d$. Let $A_i \prec_R B, 1 \leq i \leq N$ denote the random Bernoulli event that A_i dominates B w.r.t. R. Let $P_{LB}(A_i \prec_R B)$ ($P_{UB}(A_i \prec_R B)$) be a lower (upper) bound approximation of the probabilistic event $X[A_i \prec_R B] = 1$.

A lower bound of the probability $PDC(\mathcal{D}, B, R) = k$ can be derived using the following generating function:

$$\mathcal{F}^N = \prod_{i \in 1,\ldots,N} [(P_{LB}(A_i \prec_R B)) \cdot x + (1 - P_{UB}(A_i \prec_R B))] = \sum_{i \geq 0} c_i x^i. \tag{9.2}$$

Intuitively, this generating function uses the progressive approximation $P_{LB}(A_i \prec_R B)$ of the probability that A_i dominates B w.r.t. R and the progressive approximation $1 - P_{UB}(A_i \prec_R B)$ of the probability that A_i does not dominate B w.r.t. R. This generating function is equal to the uncertain generating function (cf. Equation 9.1) if the uncertain percentage (i.e. the coefficient of y) is omitted for each candidate A_i, i.e. if the coefficient of each y is set to 0.

Lemma 9.5. Let c_k be the coefficients obtained by the generating function in Equation 9.2 and let $P_{LB}(PDC(\mathcal{D}, B, R) = k)$ be the lower bound derived by applying the generating function in Equation 9.1 and exploiting Lemma 9.4. It holds that

$$c_k = P_{LB}(PDC(\mathcal{D}, B, R) = k),$$

i.e. the lower bound obtained by the (non-uncertain) generating function in Equation 9.2 is as good as the lower bound obtained using the technique in Section 9.3.3.

Proof. The lower bound derived using the uncertain generating function for the probability $PDC(\mathcal{D}, B, R) = k$ is equal to the coefficient $c_{k,0}$. The coefficient $c_{k,0}$ corresponds to the variable $x^k y^0 = x^k$. Since Equation 9.2 and Equation 9.1 are identical except for the variables containing at least one y, but no y is contained in the variable of the coefficient $c_{k,0}$ in Equation 9.1, it is identical to the coefficient c_k in Equation 9.2. □

We can see that we can use a (non-uncertain) generating function to derive the same lower bound. The advantage here is that the (non-uncertain) generating function is easier to compute, due to a much (linear in k) lower number of coefficients. However, deriving an upper bound of the probability $PDC(\mathcal{D}, B, R) = k$ is not as easy, because the upper bound does use the uncertainty of objects. An upper bound can be derived using the following generating function:

$$\mathcal{F}^N = \prod_{i \in 1,\ldots,N} [(P_{UB}(A_i \prec_R B) \cdot x + (1 - P_{LB}(A_i \prec_R B))] = \sum_{i \geq 0} c_i x^i. \tag{9.3}$$

The idea is to use a conservative approximation $P_{UB}(A_i \prec_R B)$ for the probability that A_i dominates B and a conservative approximation $1 - P_{LB}(A_i \prec_R B)$ for the probability that A_i does not dominate B. The difference of this generating function compared to the uncertain generating function (cf. Equation 9.1) is that the uncertain percentage (the coefficient of y) is added to the probabilities of both events $A_i \prec_R B = 1$ and $A_i \prec_R B = 0$, respectively[2].

---

[2] Note that adding the coefficient of y to only one of the terms of the sum will result in incorrect bounds.

Lemma 9.6. Let c_k be the coefficients obtained by the generating function in Equation 9.3 and let $P_{LB}(PDC(\mathcal{D}, B, R) = k)$ be the lower bound derived by applying the generating function in Equation 9.1 and exploiting Lemma 9.4. It holds that

$$c_k \geq P_{LB}(PDC(\mathcal{D}, B, R) = k),$$

i.e. the upper bound obtained by the (non-uncertain) generating function in Equation 9.2 is in general not as good as the lower bound obtained using the technique in Section 9.3.3.

Proof. We give an example where the upper bound derived by Equation 9.2 is worse than the upper bound derived by Equation 9.1 and exploiting Lemma 9.4. Assume a database containing uncertain objects A_1, A_2, B and R. The task is to determine the probability that $PDC(\mathcal{D}, B, R) = 1$. Also assume that we have determined (e.g. as proposed in Section 9.1) a lower bound $P_{LB}(A_1 \prec_R B)$ ($P_{LB}(A_2 \prec_R B)$) and an upper bound $P_{UB}(A_1 \prec_R B)$ ($P_{UB}(A_2 \prec_R B)$) of the probability that A_1 (A_2) dominates B w.r.t. R. To ease the notation, let A_i^+ denote $P_{LB}(A_i \prec_R B)$, let A_i^- denote $1 - P_{UB}(A_i \prec_R B)$ and let $A_i^?$ denote the uncertain fraction $P_{UB}(A_i \prec_R B) - P_{LB}(A_i \prec_R B)$. Using this notation, Equation 9.1 becomes:

$$(A_1^+ x + A_1^? y + A_1^-) \cdot (A_2^+ x + A_2^? y + A_2^-)$$

Expansion yields:

$$A_1^+ A_2^+ x^2 + A_1^+ A_2^? xy + A_1^+ A_2^- x + A_1^? A_2^+ xy + A_1^? A_2^? y^2 +$$
$$A_1^? A_2^- y + A_1^- A_2^+ x + A_1^- A_2^? y + A_1^- A_2^-$$

Exploiting Lemma 9.4 yields the following upper bound probability for $PDC(\mathcal{D}, B, R) = 1$:

$$P_{UB}(PDC(\mathcal{D}, B, R) = 1) =$$
$$A_1^+ A_2^? + A_1^+ A_2^- + A_1^? A_2^+ + A_1^? A_2^? + A_1^? A_2^- + A_1^- A_2^+ + A_1^- A_2^?$$

On the other hand, Equation 9.2 becomes:

$$((A_1^+ + A_1^?)x + A_1^- + A_1^?) \cdot ((A_2^+ + A_2^?)x + A_2^- + A_2^?)$$

Expansion yields:

$$(A_1^+ + A_1^?) \cdot (A_2^+ + A_2^?)x^2 + (A_1^+ + A_1^?) \cdot (A_2^- + A_2^?)x +$$
$$(A_1^- + A_1^?) \cdot (A_2^+ + A_2^?)x + (A_1^- + A_1^?) \cdot (A_2^- + A_2^?)$$

Extracting the coefficient c_1 of x^1 yields:

$$c_1 = (A_1^+ + A_1^?) \cdot (A_2^- + A_2^?) + (A_1^- + A_1^?) \cdot (A_2^+ + A_2^?)$$

Expansion yields:

$$c_1 = A_1^+ A_2^- + A_1^+ A_2^? + A_1^? A_2^- + A_1^? A_2^? +$$
$$A_1^- A_2^+ + A_1^- A_2^? + A_1^? A_2^+ + A_1^? A_2^?$$

Comparing $P_{UB}(PDC(\mathcal{D},B,R)=1)$ and c_1, we obtain:

$$c_1 - P_{UB}(PDC(\mathcal{D},B,R)=1) = A_1^? A_2^?$$

Thus, the upper bound c_1 of the (non-uncertain) generating function is greater (and thus worse) than the upper bound $P_{UB}(PDC(\mathcal{D},B,R)=1)$ of the uncertain generating function by $A_1^? A_2^?$. □

Intuitively, the problem of the upper bound using the (non-uncertain) generating function is that the uncertain fraction (i.e. $A_1^?$) is added to both the probability that A_i dominates B (i.e. A_1^+) and to the probability that A_i does not dominate B (i.e. A_1^-). Therefore, this approach incorrectly considers some possible worlds more often than once.

9.3.5 Efficient Domination Count Approximation Based on Disjunctive Worlds

Since the uncertain objects B and R appear in each domination relation $A_1 \prec_R B, ..., A_C \prec_R B$ that is to evaluate, we cannot split objects B and R independently (cf. Section 8.4.2). The reason for this dependency is that knowledge about the predicate $A_i \prec_R B$ may impose constraints on the position of B and R. Thus, for a partition $B_1 \subset B$, the probability $P(A_j \prec_R B_1)$ may change given $A_i \prec_R B$ $(1 \leq i,j \leq C, i \neq j)$. However, note:

Lemma 9.7. Given fixed partitions $B' \subseteq B$ and $R' \subseteq R$, then the random variables $A_i \prec_{R'} B'$ are mutually independent for $1 \leq i,j \leq C, i \neq j$.

Proof. Similar to the proof of Lemma 9.3. □

This allows us to individually consider the subset of possible worlds where $b \in B'$ and $r \in R'$ and use Lemma 9.7 to efficiently compute the approximated domination count probabilities $PDC_{LB}^k(\mathcal{D},B',R')$ and $PDC_{UB}^k(\mathcal{D},B',R')$ under the condition that B falls into a partition $B' \subseteq B$ and R falls into a partition $R' \subseteq R$. This can be performed for each pair $(B',R') \in \underline{\mathcal{B}} \times \underline{\mathcal{R}}$, where $\underline{\mathcal{B}}$ and $\underline{\mathcal{R}}$ denote the decompositions of B and R, respectively. Now, we can treat pairs of partitions $(B',R') \in \underline{\mathcal{B}} \times \underline{\mathcal{R}}$ independently, since all pairs of partition represent disjunctive sets of possible worlds due to the assumption of a disjunctive partitioning. Exploiting this independency, the PDF of the domination count $PDC(\mathcal{D},B,R)$ of the total objects B and R can then be obtained by creating an uncertain generating function for each pair (B',R') to derive a lower and an upper bound of $P(PDC(\mathcal{D},B',R')=k)$ and then computing the weighted sum of these bounds as follows:

$$PDC_{LB}^k(\mathcal{D},B,R) = \sum_{B' \in \underline{\mathcal{B}}, R' \in \underline{\mathcal{R}}} PDC_{LB}^k(\mathcal{D},B',R') \cdot P(B') \cdot P(R').$$

The complete algorithm of our domination count approximation approach can be found in the next Section.

Algorithm 7 PDC($\mathcal{D}$, B, R)

```
$influenceObjects = \emptyset$
$CompleteDominationCount = 0$
$//Complete\ Domination$
for all $A_i \in \mathcal{D}$ do
    if DDC$_{OPT}(A_i, B, R)$ then
        $CompleteDominationCount{+}{+}$
    else if $\neg$DDC$_{OPT}(B, A_i, R)$ then
        $influenceObjects = influenceObjects \cup \{A_i\}$
    end if
end for
$//probabilistic\ domination\ count$
$PDC_{LB}$= [0,...,0] //length $|\mathcal{D}|$
$PDC_{UB}$= [1,...,1] //length $|\mathcal{D}|$
while $\neg$ stopcriterion do
    split($R$), split($B$), split($A_i \in \mathcal{D}$)
    for all $B' \in B$, $R' \in R$ do
        $P_{LB}$= [0,...,0] //length $|influenceObjects|$
        $P_{UB}$= [1,...,1] //length $|influenceObjects|$
        for all $(0 < i < |influenceObjects|)$ do
            $A_i = influenceObjects[i]$
            for all $A'_i \in A_i$ do
                if DDC$_{OPT}(A'_i, B', R')$ then
                    $P_{LB}[i] = P_{LB}[i] + P(A'_i)$
                else if DDC$_{OPT}(B', A'_i, R')$ then
                    $P_{UB}[i] = P_{UB}[i] + P(A'_i)$
                end if
            end for
        end for
        compute $PDC_{LB}(\mathcal{D}, B', R')$ and $PDC_{UB}(\mathcal{D}, B', R')$ based on $P_{LB}$ and $P_{UB}$ using
        UGFs.
        for all $(0 < i < \mathcal{D})$ do
            $PDC_{LB}[i] = PDC_{LB}[i] + PDC_{LB}(\mathcal{D}, B', R') \cdot P(B') \cdot P(R')$
            $PDC_{UB}[i] = PDC_{UB}[i] + PDC_{UB}(\mathcal{D}, B', R') \cdot P(B') \cdot P(R')$
        end for
    end for
    ShiftRight($PDC_{LB}$,$CompleteDominationCount$)
    ShiftRight($PDC_{UB}$,$CompleteDominationCount$)
end while
return ($PDC_{LB}$, $PDC_{UB}$)
```

9.4 Implementation

Algorithm 7 is a complete method for iteratively computing and refining the probabilistic domination count for a given object B and a reference object R. The algorithm starts by detecting complete domination (cf. Section 9.2.1). For each object that completely dominates B, a counter *CompleteDominationCount* is increased and each object that is completely dominated by B is removed from further consideration, since it has no influence on the domination count of B. The remaining objects, which may have a probability greater than zero and less than one to dominate B, are stored in a set $influenceObjects$. The set $influenceObjects$ is now used to compute the probabilistic domination count (PDC_{LB}, PDC_{UB}). PDC_{LB} and PDC_{UB} are lists containing, at each position i, a lower and an upper bound for $P(PDC(\mathcal{D}, B, R) = i)$, respectively. This notation is equivalent to a single uncertain domination count PDF. Thereby we reduce the number of considered objects for bounding the probabilistic domination count to the $influenceObjects$ only. At the end of each iteration we then use the $ShiftRight()$ operation which shifts the domination count PDF to the right by *CompleteDominationCount* positions. The main loop of the probabilistic domination count approximation starts in line 14. In each iteration, B, R, and all influence objects are split into an increasing number of disjunct partitions. For each combination of partitions B' and R', and each database object $A_i \in influenceObjects$, the probability $P(A_i \prec_{R'} B')$ is approximated (cf. Section 9.1). These domination probability bounds are used to build an uncertain generating function (cf. Section 9.3.3) for the domination count of B' w.r.t. R'. Finally, these domination counts are aggregated for each pair of partitions B', R' into the domination count $PDC(\mathcal{D}, B, R)$ (cf. Section 9.3.5). The main loop continues until a domain- and user-specific stop criterion is satisfied. For example, for a threshold kNN query, a stop criterion is to decide whether the lower (upper) bound that B has a domination count of less than (at least) k, exceeds (falls below) the given threshold.

The progressive decomposition of objects (line 15) can be facilitated by precomputed split points at the object PDFs. More specifically, we can iteratively split each object X by means of a median-split-based bisection method and use a kd-tree [24] to hierarchically organize the resulting partitions. In other words each call of the split(X)-method provides us with the partitioning corresponding to the next level of the kd-tree. The kd-tree is a binary tree. The root of a kd-tree represents the complete region of an uncertain object. Every node implicitly generates a splitting hyperplane that divides the space into two subspaces. This hyperplane is perpendicular to a chosen split axis and located at the median of the node's distribution in this axis. The advantage is that, for each node in the kd-tree, the probability of the respective subregion X' is simply given by $0.5^{X'.level-1}$, where $X'.level$ is the level of X'. In addition, the bounds of a subregion X' can be determined by backtracking to the root. In general, for continuously partitioned uncertain objects, the corresponding kd-tree may have an infinite height, however for practical reasons, the height h of the kd-tree is limited. The choice of h is a trade-off between approximation quality and efficiency: for a very large h, considering each leaf node is similar to applying integration on the PDFs, which yields an exact result; however, the number of leaf nodes, and thus the worst case complexity increases exponentially in h. Note that our experiments (c.f. Section 9.6) show that a low h value is sufficient to yield reasonably tight approximation bounds. Yet it has to be noted, that in the general case of continuous uncertainty, our proposed approach may only return an approximation of the exact probabilistic domination count. However, such an approximation may be sufficient to decide a given predicate as we will see in Section 9.5 and even in the case where the approximation does not suffice to decide the query predicate, the approximation will give the user a confidence value, based on which a user may be able decide whether to include an object in the result.

9.5 Applications

In this section, we outline how the probabilistic domination count can be used to efficiently evaluate or accelerate a variety of probabilistic similarity query types, namely the probabilistic inverse similarity ranking query [111], the probabilistic threshold k-NN query [51], the probabilistic threshold reverse k-NN query and the probabilistic similarity ranking query [33, 57, 108, 146]. We start with the probabilistic inverse ranking query, because it can be derived trivially from the probabilistic domination count introduced in Section 8.4.2. In the following, let $\mathcal{D} = \{A_1, ..., A_N\}$ be an uncertain database containing uncertain objects $A_1, ..., A_N$.

Corollary 9.1. Let B and R be uncertain objects. The task is to determine the probabilistic ranking distribution $Rank(\mathcal{D}, B, R)$ of B w.r.t. similarity to R, i.e. the distribution of the position $Rank(\mathcal{D}, B, R)$ of object B in a complete similarity ranking of $A_1, ..., A_N, B$ w.r.t. the distance to an uncertain reference object R. Using our techniques, we can compute $Rank(\mathcal{D}, B, R)$ as follows:

$$P(Rank(\mathcal{D}, B, R) = i) = P(PDC(\mathcal{D}, B, R) = i - 1)$$

The above corollary is evident, since the proposition "B has rank i" is equivalent to the proposition "B is dominated by exactly $i - 1$ objects".

The most prominent probabilistic similarity search query is the probabilistic threshold kNN query.

Corollary 9.2. Let $Q = R$ be an uncertain query object and let k be a scalar. The problem is to find all uncertain objects $PkNN(Q, \mathcal{D}, \tau)$ that are the k-nearest neighbors of Q with a probability of at least τ. Using our techniques, we can compute the probability $P(B \in kNN(Q, \mathcal{D}))$ that an object B is a kNN of Q as follows:

$$P(B \in kNN(Q, \mathcal{D})) = \sum_{i=0}^{k-1} P(PDC(\mathcal{D}, B, Q) = i)$$

The above corollary is evident, since the proposition "B is a kNN of Q" is equivalent to the proposition "B is dominated by less than k objects". To decide whether B is a kNN of Q, i.e. if $B \in PkNN(Q, \mathcal{D}, \tau)$, we just need to check if $P(B \in kNN(Q, \mathcal{D})) > \tau$.

Next we show how to answer probabilistic threshold RkNN queries.

Corollary 9.3. Let $Q = R$ be an uncertain query object and let k be a scalar. A probabilistic reverse k-nearest neighbour (PRkNN) query finds all uncertain objects A_i that have Q as one of their kNNs with a probability of at least τ, that is, all objects A_i for which it holds that $Q \in PkNN(Q, \mathcal{D}, \tau)$. Using our techniques, we can compute the probability $P(B \in RkNN(Q, \mathcal{D}))$ that an object B is an RkNN of Q as follows:

$$P(B \in RkNN(Q, \mathcal{D})) = \sum_{i=0}^{k-1} P(PDC(\mathcal{D}, Q, B) = i)$$

The intuition here is that an object B is a RkNN of Q if and only if Q is dominated less than k times w.r.t. B.

For PkNN and PRkNN queries, the total complexity to compute the uncertain generating function can be improved from $O(|influenceObjects|^3)$ to $O(|influenceObjects| \cdot k^2)$ since it can be observed from Corollaries 9.2 and 9.3 that for kNN and RkNN queries, we only require the section of the PDF of $PDC(\mathcal{D}, B, R)$ where $PDC(\mathcal{D}, B, R) < k$, i.e. we only need to know the probabilities $P(PDC(\mathcal{D}, B, R) = x), x < k$. This can be exploited to improve the runtime of the computation of the PDF of $PDC(\mathcal{D}, B, R)$ as follows: Consider the iterative computation of the generating functions $\mathcal{F}^1, ..., \mathcal{F}^{|influenceObjects|}$. For each $\mathcal{F}^l, 1 \leq l \leq |influenceObjects|$, we only need to consider the coefficients $c_{i,j}$ in the generating function $\mathcal{F}^i$ where $i < k$, since only these coefficients have an influence on $P(PDC(\mathcal{D}, B, R) = x), x < k$ (cf. Section 9.4). In addition, we can merge all coefficients $c_{i,j}$, $c_{i',j'}$ where $i = i'$, $i + j > k$ and $i' + j' > k$, since all these coefficients only differ in their influence on the upper bounds of $P(PDC(\mathcal{D}, B, R) = x), x \geq k$, and are treated equally for $P(PDC(\mathcal{D}, B, R) = x), x < k$. Thus, each $\mathcal{F}^l$ contains at most $\sum_{i=1}^{k+1} i$ coefficients (one $c_{i,j}$ for each combination of i and j where $i + j \leq k$). Thus reducing the total complexity to $O(k^2 \cdot |influenceObjects|)$.

Finally, we show how techniques can be utilized to boost applications calculating the expected rank (cf. [57]) of an uncertain object.

Corollary 9.4. Let $Q = R$ be an uncertain query object. The problem is to rank the uncertain objects A_i according to their expected rank $E(Rank(A_i))$ w.r.t. the distance to Q. The expected rank of an uncertain object A_i can be computed as follows:

$$E(Rank(A_i)) = \sum_{i=0}^{N-1} P(PDC(\mathcal{D}, Q, A_i) = i) \cdot (i + 1)$$

Other probabilistic similarity queries (e.g. kNN and RkNN queries with a different uncertainty predicate instead of a threshold τ) can be approximated efficiently using our techniques as well. Details are omitted due to space constraints.

9.6 Experimental Evaluation

In this section, we review the characteristics of the proposed algorithm on synthetic and real-world data. The algorithm will be referred to as **IDCA** (Iterative Domination Count

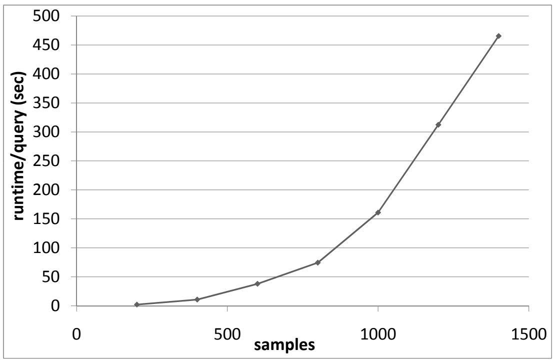


Figure 9.3: Runtime of **MC** for increasing sample size.

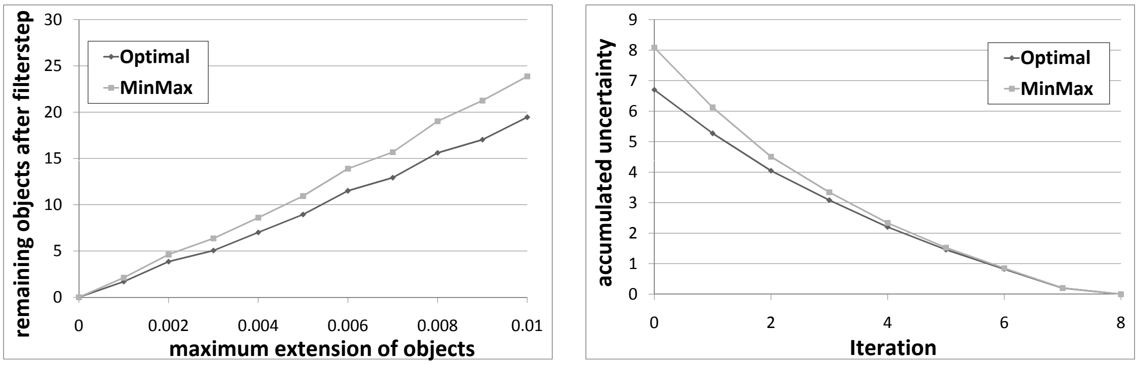
Approximation). We performed experiments under various parameter settings. Unless otherwise stated, for 100 queries, we chose B to be the object with the 10^{th} smallest MinDist to the reference object R. We used a synthetic dataset with 10,000 objects modeled as 2D rectangles in $[0,1]^2$. The degree of uncertainty of the objects in each dimension is modeled by their relative extent. The extents were generated uniformly and at random with 0.004 as maximum value. For the evaluation on real-world data, we utilized the International Ice Patrol (IIP) Iceberg Sightings Dataset[3]. This dataset contains information about iceberg activity in the North Atlantic in 2009. The latitude and longitude values of sighted icebergs serve as certain 2D mean values for the 6,216 probabilistic objects that we generated. Based on the date and the time of the latest sighting, we added Gaussian noise to each object, such that the passed time period since the latest date of sighting corresponds to the degree of uncertainty (i.e. the extent). The extents were normalized w.r.t. the extent of the data space into the $[0,1]^2$-space, and the maximum extent of an object in either dimension is 0.0004.

9.6.1 Runtime of the Monte-Carlo-based Approach

To the best of our knowledge, there exists no approach which is able to process uncertain similarity queries on probabilistic databases with continuous PDFs. A naive approach needs to consider all possible worlds and thus needs to integrate over all object PDFs, implying a runtime exponentially in the number of objects. Since this is not applicable even for small databases, we adapted an existing approach to cope with the conditions. The approach most related to our work is [111], which solves the problem of computing the domination count for a certain query and discrete distributions within the database objects. Thus the proposed comparison partner works as follows: Draw a sufficiently large number S of samples from each object by Monte-Carlo-Sampling. Then, for each sample $q_i \in Q$ of the query, apply the algorithm proposed in [111] to compute an exact probabilistic domination count PDF of an object B. As proposed in [111], this is done using the generating function

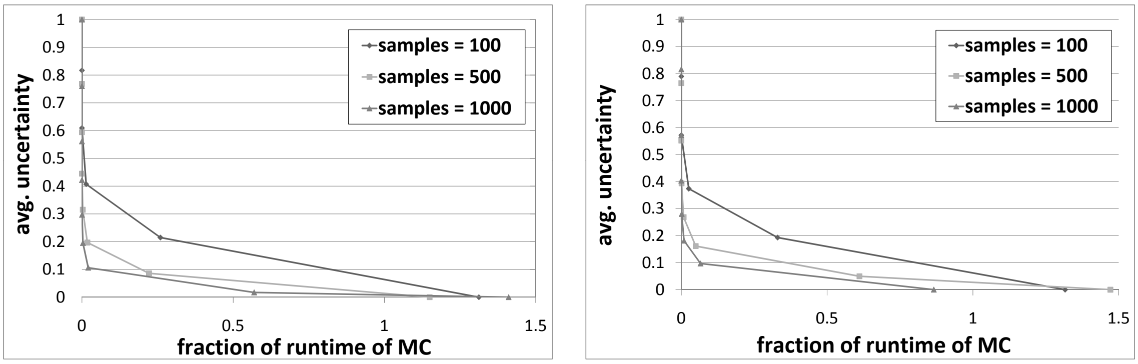
---

[3] The IIP dataset is available at the National Snow and Ice Data Center (NSIDC) web site (*http://nsidc.org/data/g00807.html*).



(a) Candidates after spatial pruning. (b) Accumulated uncertainty of result.

Figure 9.4: Optimal vs. MinMax decision criterion.



(a) Synthetic Data (b) Real Data

Figure 9.5: Uncertainty of **IDCA** w.r.t. the relative runtime to **MC**.

technique and using an *and/xor tree* to combine individual samples into discrete distributed uncertain objects. Finally, accumulate the resulting certain domination count PDFs of each $q_i \in Q$ into a single domination count PDF by taking the average. The execution time for this approach, which we will refer to as **MC** in the following, is shown in Figure 9.3. It can be observed that for a reasonable sample size (which is required to achieve a result that is close to the correct result with high probability) the runtime becomes very large.

Note that our comparison partner only works for discrete uncertain data. To make a fair comparison our approach relies on the same uncertainty model (default: 1000 samples/object). Nevertheless, all the experiments yield analogous results for continuous distributions.

9.6.2 Optimal vs. MinMax Domination Decision Criterion

In the first experiment, we evaluate the gain of pruning power using the complete similarity domination technique (cf. Section 9.2.1) instead of the MinMax domination decision

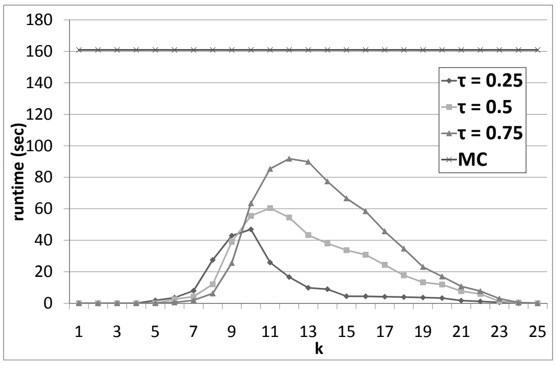


Figure 9.6: Runtimes of **IDCA** and **MC** for different query predicates k and τ.

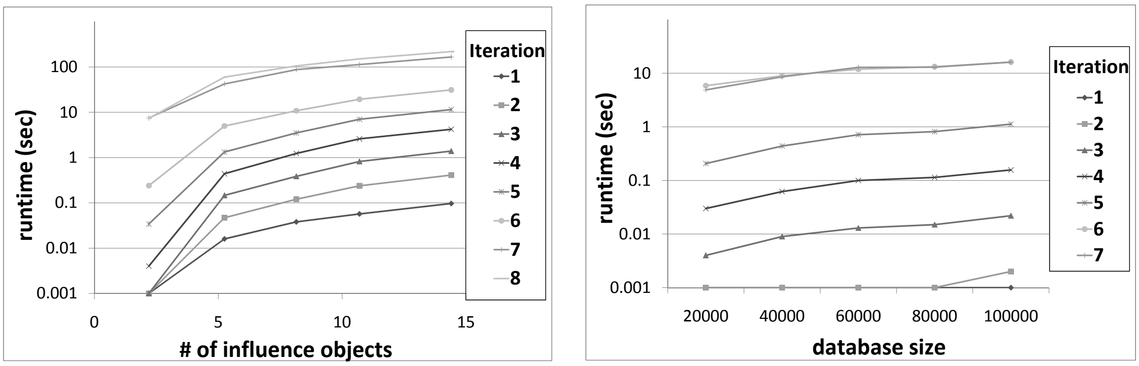
criterion which is still often used in many probabilistic query processing pipelines to prune uncertain objects from the search space. The first experiment evaluates the number of uncertain objects that cannot be pruned using complete domination only, that is the number of candidates are to evaluate in our algorithm. Figure 9.4(a) shows that the optimal domination decision criterion (DDC$_{OPT}$) is able to prune about 20% more candidates than the MinMax (DDC$_{MM}$) criterion. In addition, we evaluated the domination count approximation quality (in the remainder denoted as uncertainty) after each decomposition iteration of the algorithm, which is defined as the sum $\sum_{i=0}^{N} PDC^i_{UB}(\mathcal{D}, B, R) - PDC^i_{LB}(\mathcal{D}, B, R)$. The result is shown in Figure 9.4(b). The improvement of the complete domination (denoted as iteration 0) can also be observed in further iterations. After enough iterations, the uncertainty converges to zero for both approaches.

9.6.3 Iterative Domination Count Approximation

Next, we evaluate the trade-off of our approach regarding approximation quality and the invested runtime of our domination count approximation. The results can be seen in Figure 9.5 for different sample sizes and datasets. It can be seen that initially, i.e. in the first iterations, the average approximation quality (avg. uncertainty of an *influenceObject*) decreases rapidly. The less uncertainty left, the more computational power is required to reduce it any further. Except for the last iteration (resulting in 0 uncertainty) each of the previous iterations is considerably faster than **MC**. In some cases (see Figure 9.5(b)) **IDCA** is even faster in computing the exact result.

9.6.4 Queries with a Predicate

Integrated in an application one often wants to decide whether an object satisfies a predicate with a certain probability. In the next experiment, we posed queries in the form: Is object B among the k nearest neighbors of Q (predicate) with a probability of 25%, 50%, 75%? The results are shown in Figure 9.6 for various k-values. With a given predicate, **IDCA** is often able to terminate the iterative refinement of the objects earlier in most of



(a) Runtime w.r.t. number of influence objects. (b) Runtime for different sizes of the database.

Figure 9.7: Impact of influencing objects.

the cases, which results in a runtime which is orders of magnitude below **MC**. In average the runtime is below **MC** in all settings.

9.6.5 Number of influenceObjects

The runtime of the algorithm is mainly dependent on the number of objects which are responsible for the uncertainty of the rank of B. The number of *influenceObjects* depends on the number of objects in the database, the extension of the objects and the distance between Q and B. The larger this distance, the higher the number of *influenceObjects*. For the experiments in Figure 9.7(a) we varied the distance between Q and B and measured the runtime for each iteration. In Figure 9.7(b) we present runtimes for different sizes of the database. The maximum extent of the objects was set to 0.002 and the number of objects in the database was scaled from 20,000 to 100,000. Both experiments show that **IDCA** scales well with the number of influencing objects.

9.7 Conclusions

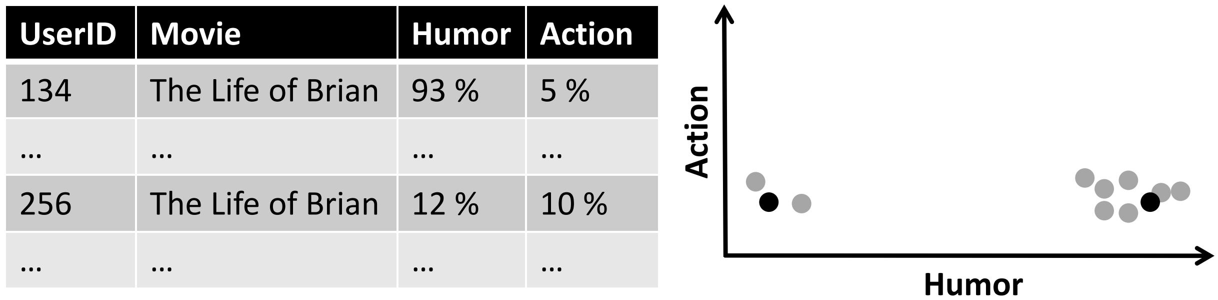
In this chapter, we transferred the concept of spatial domination to the domain of uncertain data. We utilized the optimal domination decision criteria as a filter to conservatively and progressively approximate the probability that an object is being dominated by another object. An iterative filter-refinement strategy is used to stepwise improve this approximation in an efficient way. Specifically we propose a method to efficiently and effectively approximate the probabilistic domination count of an object using the novel technique of uncertain generating functions. We showed that the proposed concepts can be used as a probabilistic pruning criterion to efficiently answer a wide range of probabilistic similarity queries while keeping correctness according to the possible world semantics. Our experiments show that our iterative filter-refinement strategy is able to achieve a high level of precision at a low runtime. The question at hand is how much are these techniques able to improve similarity search algorithms on uncertain data.

Chapter 10

Probabilistic Reverse Nearest Neighbour Queries on Discrete Uncertain Data

In this chapter we want to show how the techniques for approximating the probabilistic domination count can utilized in order to develop efficient similarity algorithms on uncertain data. Specifically we will focus on probabilistic reverse nearest neighbor (PRNN) queries. Given a query object q, a reverse nearest neighbor query in a common certain database returns the objects having q as their nearest neighbor. PRNN queries consequently return the uncertain objects having the query object as nearest neighbor with a sufficiently high probability. We propose an algorithm for efficiently answering PRNN queries using new pruning mechanisms based on probabilistic domination. We compare our algorithm to state-of-the-art approaches recently proposed. Our experimental evaluation shows that our approach is able to significantly outperform previous approaches. In addition, we show how our approach can easily be extended to PRkNN (where $k > 1$) query processing for which there is currently no efficient solution. Parts of this chapter have been published in [31].

The rest of this chapter is organized as follows: First we give an overview of sample applications in Section 10.1 and a formal definition of the problem of PRNN queries in Section 10.2. In 10.3, we describe the general framework for PRNN query processing. In Section 10.4, we give a description including implementation details of our proposed algorithm. In Section 10.5 we show how to extend the PRNN framework and algorithm to efficiently answer PRkNN queries. We will subsequently show how the two existing approaches for PRNN query processing fit in this framework (cf Section 10.6). All proposed techniques and the competitors are experimentally evaluated in Section 10.7. Finally Section 10.8 concludes this chapter.



UserID	Movie	Humor	Action
134	The Life of Brian	93 %	5 %
…	…	…	…
256	The Life of Brian	12 %	10 %
…	…	…	…

Figure 10.1: Uncertain object example: user ratings.

10.1 Applications of PRkNN Queries

RkNN query processing has been studied extensively on certain data (cf Section 2.4). However, due to the immense number of applications dealing with uncertain data, novel solutions to cope with uncertain objects are required. The main challenge here is that the event that an object belongs to an RkNN result set, is no longer a predicate, but a random variable that may be true with some probability. In this chapter, we study the problem of probabilistic reverse k-nearest neighbor (PRkNN) search in uncertain databases. A PRkNN query returns the set of objects having a sufficiently high probability to be the reverse k-nearest neighbor of a query object. Let us note that the query object can be uncertain as well. Traditional methods as proposed in [148, 152] do not qualify for uncertain objects, since an uncertain object – instead of being a single point in a multi-dimensional space – is a random variable defined by the probability distribution over the distance space. According to the possible worlds model [1], an uncertain database can be viewed as a set of possible database instances (worlds), to which traditional pruning methods can be applied to. Since the number of possible worlds is exponential in the number of objects, we need special methods to avoid consideration of all possible worlds.

There is a wide field of applications for PRNN queries (k=1), e.g. decision support, marketing, location-based services among others [45, 110]. For instance consider a movie recommendation database, where each movie is represented by the set of reviews for this movie, each consisting of a set of attribute values. Examples of such attributes are classification of the genre, humoristic value and suspense. An example for two such records is given in Figure 10.1. Thus, each movie record is represented by multiple records from different users. Differences between records of the same movie reflect the uncertainty in the user ratings. The advantage of using this uncertain data instead of simply using the average user recommendation of each movie record is shown by the following example: Consider a movie like “Monty Python’s The Life of Brian” [44]: Many users will rate this movie as extremely funny. However, since this movie is based on a rather black sense of humor, some users may rate this movie as absolutely not funny. Thus, this movie would have an average of “moderately funny” and would result in a very large distance to other funny movies. Thus, this movie would never be recommended to users purchasing funny movies, even though there is a high probability that a user looking for funny movies may indeed be interested in this movie. For recommendation purposes a system now wants to report a KNN-list (consisting of the probabilistic k-nearest neighbours) of movies that are similar to a movie the user is interested in. For performance reasons these KNN-lists are precomputed for each movie in the database. Whenever a new movie is added to the database this may cause an update of various KNN-lists. A naive approach would thus recompute the KNN-lists of all movies which would result in a large computational overhead. A better approach however would be to specifically update the KNN-lists of movies which are among the probabilistic reverse k-nearest neighbours of the newly inserted object. This is usually a much smaller set and thus much more efficient. Additionally, probabilistic RNN queries also have applications in the following fields:

- In privacy preserving location-based services, the exact location of every user is obfuscated into a cloaked spatial region [119]. Yet, users may still be interested in finding their RNNs.
- In stock market trend analysis, a stock s has many deals. A deal is recorded by price (per share) and the volume (number of shares). In such an application, we can model stocks as uncertain objects treating its deals as uncertain instances. For a given stock s, clients may be interested in finding all other stocks that have trading trends more similar to s than others. [45].
- Consider a density based clustering on uncertain data based on nearest-neighbor distances, e.g. single link clustering. When an object is inserted, removed or updated a naive approach would require the whole clustering to be recomputed. However, we can exploit that only the RNNs of the inserted/removed/updated object are affected.

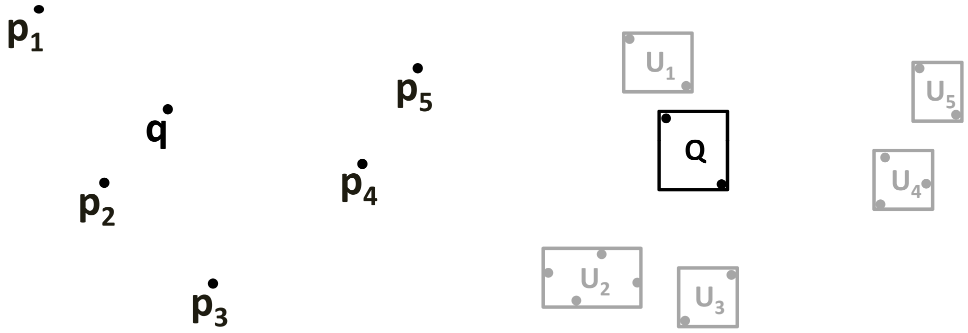
There is a large number of other applications for queries that consider the proximity of uncertain objects (e.g. [97, 50, 35]) for which the applications of RNN queries on uncertain objects are very similar.

10.2 Problem Definition

Traditional (monochromatic) RNN queries have already been defined in Definition 2.5. An example is illustrated in Figure 10.2(a). Consider the point object set ($p_{1-5} \in P$) with q as query object. Here, the result of an RNN query would contain the objects p_1 and p_2. The uncertain case however is not that trivial (cf Figure 10.2(b)).

10.2.1 Uncertainty Model

Following the Block-Independent Disjoint Scheme ([106]) which is one of the most commonly used uncertainty models, we assume the following: a probabilistic database $\mathcal{D}$ is given by a set of uncertain objects $\mathcal{D} = \{U_1, \ldots, U_n\}$ with d uncertain attributes. An uncertain object U_i is represented by a set of d-dimensional points $u_1, \ldots, u_m$ reflecting all



(a) RNN example. (b) PRNN example.

Figure 10.2: Examples for RNN and PRNN.

possible instances of U_i. Each instance u_j is assigned with a probability $P(u_j)$ denoting the probability that U_i appears at u_j, i.e. all instances of U_i reflect the probability distribution of U_i. The probability distributions of each two objects are pairwise independent and the events of occurrence of all instances $u \in U_i$ are mutually exclusive. For clarity we assume $P(U_i) = \Sigma_{j=1}^{m} u_j = 1$, although the proposed algorithms can easily be adapted to the case where $P(U_i) \leq 1$. A possible world $W = u_1, \ldots, u_n$ is a set of instances containing one instance from each object and occurring with an appearance probability of $P(W) = \Pi_{i=1}^{n} P(u_i)$. Let Ω denote the set of all possible worlds, then $\Sigma_{W \in \Omega} P(W) = 1$.

10.2.2 PRNN Queries in Uncertain Databases

For PRNN queries on uncertain databases the threshold parameter τ is introduced (cf. [45, 110]. Using τ as threshold, the user can restrict the result set to objects which have at least a predefined probability to be in the result in order to avoid reporting unnecessarily many results that are unlikely. A probabilistic reverse nearest neighbor query $PRNN_Q^\tau$ then returns the set of all objects $U_i \in \mathcal{D}$ where $P(U_i \in RNN_Q)$ (in the following denoted by $P(RNN_Q(U_i))) \geq \tau$. Naively, this probability can be calculated by performing a (non-probabilistic) RNN query on each possible world:

$$P(RNN_Q(U_i)) = \sum_{W \in \Omega} P(W) \cdot \delta(RNN_Q^W(U_i))$$

where $\delta(RNN_Q^W(U_i))$ is an indicator function that is 1 if U_i is a reverse nearest neighbor of Q in world W and 0 otherwise. In the example given in Figure 10.2(b), $RNN_Q(U_1)$ holds in all possible worlds, therefore $P(RNN_Q(U_1)) = 1$. In contrast, $P(RNN_Q(U_2)) = P(RNN_Q(U_4)) = P(RNN_Q(U_5)) = 0$, since there is no possible world in which $RNN_Q(U_2)$, $RNN_Q(U_4)$ or $RNN_Q(U_5)$ hold, as in each possible world the nearest neighbor of U_2 is U_1 and the nearest neighbor of U_4 is U_5 and vice versa. For U_3, we obtain the probability $P(RNN_Q(U_3))$ by building the sum of the probabilities of all possible worlds where at least one of the objects U_i ($i \in \{1, 2, 4, 5\}$) is closer to U_3 than Q to U_3.

Obviously, this brute-force approach taking each possible world into account is in general not applicable because the number of possible worlds grows exponentially with the number of involved uncertain objects.

As we will see PRNN queries are CPU bound (in the presence of an appropriate index structure) and we have to concentrate on optimizing the CPU operations. To the best of our knowledge, there are currently two approaches for answering PRNN queries. The approach from Chen et al. [110] which is designed for PRNN queries on uncertain objects represented by continuous probability density functions (PDFs) and the approach from Cheema et al. [45] which works for the discrete case only. Both algorithms are discussed in more detail in Section 10.6.

In order to reduce the computational overhead of such queries, in this chapter we introduce efficient filter methods used to exclude (prune) as many objects as possible from the expensive query evaluation process.

10.3 PRNN Algorithm Sketch

In this section, we propose our approach for PRNN processing. For this purpose we follow a multi-step framework consisting of the following four steps:

- Approximation: This step happens prior to the actual query processing and includes offline approximations and precomputations which can be used for boosting the query process.
- Spatial Pruning: During query processing spatial pruning is used to prune objects which cannot be part of the result independent of the threshold parameter τ. Therefore only the most basic approximation of the objects is usually utilized.
- Probabilistic Pruning: In this step the remaining objects after spatial pruning are examined. For each of these candidates a bound for the candidate to belong to the result is calculated. If the lower (upper) bound of this probability is larger (smaller) than the threshold τ then the candidate is included in (excluded from) the result and is not further considered.
- Verification: The remaining objects after the probabilistic pruning step are then verified in a usually expensive verification step. This step yields the final result.

In the following we will give a more detailed description of the above four steps and discuss how our proposed algorithm fits into this framework. The detailed implementation of our PRNN query processing approach can be found in Section 10.4. The above framework will also be used for detailed evaluation of the comparison partners. Therefore we show how the two existing solutions by Cheema et al. [45] (in the following called CLWZP) and by Lian et al. [110] (in the following called LC) are implemented according to this framework in Section 10.6.

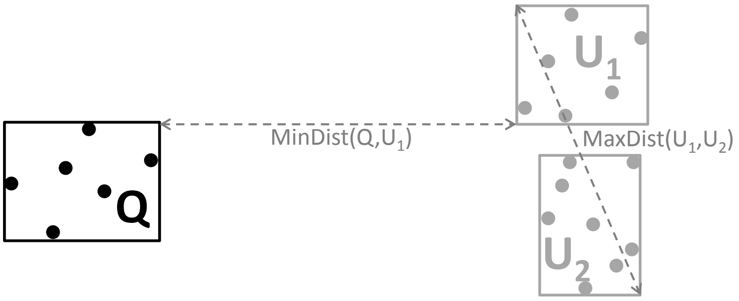


Figure 10.3: Pruning uncertain objects using minimal and maximal distance.

10.3.1 Approximation of Objects

The probability distribution (or more specifically the uncertainty region) assigned to an uncertain object can become arbitrarily complex causing expensive distance computations at query time. A common solution to overcome this problem is to use conservative approximations, like spheres or rectangles providing efficient distance computation in a filter step. Similar to the approximation technique used for the CLWZP algorithm ([45]), in order to approximate uncertain objects we use minimum bounding boxes covering the uncertainty regions. This approximation is mainly used for spatial pruning. For probabilistic pruning, we need a more detailed object approximation. Therefore, we additionally assume that each uncertainty region of an object $X \in \mathcal{D} \cup \{Q\}$ is hierarchically decomposed using a hierarchical space partitioning scheme. Specifically, we use an R*-tree [21] to hierarchically organize the instances of X. In addition to the spatial keys, each index entry stores the aggregated probability of all instances in the corresponding subtree. When traversing the index assigned to an uncertain object X in a breadth-first manner, each level of the R*-tree provides a disjoint and complete partitioning $\underline{\mathcal{X}}$ of X such that each partition $X' \in \underline{\mathcal{X}}$ contains a non-empty set of $m' \leq m$ instances $\{x'_1, ..., x'_{m'}\}$ and

$$\bigcup_{X' \in \underline{\mathcal{X}}} = \{x_1, ..., x_m\} \text{ and } \forall X'_i \in \underline{\mathcal{X}}, X'_{j \neq i} \in \underline{\mathcal{X}} : X'_i \cap X'_j = \emptyset.$$

An index entry representing partition X'_i contains the aggregated probability $P(X'_i) = \sum_{x'_j \in X'_i} P(x'_j)$. Note that the disjoint property implies that any instance $x \in X$ is contained in at most one partition. The rectangular approximations of the instances contained in each R*-tree node however may overlap. Let us note that we have to carefully select the refinement resolution since the PRNN computation is CPU-bound, as we will see in our experiments (cf. Section 10.7). We obtain a good control of this variable by using a very low R*-tree node capacity, e.g. in our experiments we used less than four entries per node.

To summarize, we use a memory-resident R*-tree to organize the objects $X \in \mathcal{D}$ (global R*-Tree). The leaf entries containing the MBRs of the objects point to the local R*-trees of the objects. The local R*-trees are stored in a breadth-first manner, such that the highest levels of the trees can be obtained with one single scan.

10.3.2 Spatial Pruning

As already discussed it is possible to (spatially) prune objects without considering their probability distributions when using only the (spatial) approximations of the objects. Therefore, a pruning technique is needed. The concept of spatial domination is obviously applicable for this purpose. Thus all techniques for detecting spatial domination as reviewed and proposed in Part II can be used. For instance an object B can be pruned by an object A for a query Q if $\mathrm{DDC}_{MM}(A, B, Q)$.

Example 10.3.1. Consider the example shown in Figure 10.3. For a R1NN query, object U_1 can be excluded from further consideration, as the maximal distance between U_1 and U_2 is smaller than the minimal distance between Q and U_1. Thus U_1 can never be R1NN of Q.

The used pruning technique for uncertain objects efficiently organized by an index is easily extendable for pruning higher-level pages of the index. For example assume object U_1 in Figure 10.3 to be an index page containing several uncertain objects. Then all objects in this page can be pruned immediately.

10.3.3 Probabilistic Pruning

Probabilistic pruning is performed for objects that cannot be pruned spatially. In the probabilistic pruning step, the uncertainty regions of objects are partitioned. The aim of this partitioning is to prune more objects based on the probability threshold τ.

For this purpose we can easily adapt the techniques proposed in Section 9.2 for computing the probabilistic domination count. If for an uncertain object U it holds that $PDC^1_{LB}(\mathcal{D}, Q, U) > \tau$ then U can be probabilistically pruned. The decomposition scheme of objects is thereby given by the local R*-Tree of the objects. Whenever we want to refine this bound we may consider the next level of entries in the local indexes of all involved objects.

10.3.4 Verification

An object U_i which cannot be pruned by the pruning techniques is denoted as candidate. The next step requires each candidate to be verified, which means it has to be checked if $P(RNN_Q(U_i)) \geq \tau$. This involves finding all objects which affect this probability and considering these objects in more detail. The verification step is very expensive, since many possibilities have to be considered. In our implementation we rely on the generating function technique as proposed in [108] for computing the exact result probabilities of each candidate.

10.4 Implementation

In this section, we describe the implementation of our PRNN algorithm.

Algorithm 8 PRNN(Q, I_{DB}, τ, *depth*)

```
$S_{cnd} = \emptyset$, $S_{prn} = \emptyset$
spatialPruning($Q$, $I_{DB}$, $S_{cnd}$, $S_{prn}$)

$S_{res} = \emptyset$
for each $B \in S_{cnd}$ do
    $S_{ifl}$ = getInfluenceObjects($Q$, $B$, $S_{cnd} \setminus \{B\}$, $S_{prn}$)
    i := probabilisticPruning($Q$, $B$, $S_{ifl}$, $\tau$, depth)
    if i=1 then
        //B cannot be RNN of Q
    else if i=-1 then
        //B is RNN of Q
        $S_{res} = S_{res} \cup \{B\}$
    else
        //B has to be verified
        if verify($Q$, $B$, $S_{ifl}$, $\tau$) then
            $S_{res} = S_{res} \cup \{B\}$
        end if
    end if
end for
return $S_{res}$
```

10.4.1 Overview

Algorithm 8 combines the main modules for PRNN processing. The input requires an uncertain query object Q, an R-tree based index structure organizing the uncertain objects from the database I_{DB}, a probability threshold τ and a parameter *depth* controlling the depth of our probabilistic pruning computation. The spatialPruning method fills the sets S_{cnd} with potential result objects and S_{prn} with objects (entries) which can certainly be excluded using a technique for detecting spatial domination (cf Part II). For each remaining object $B \in S_{cnd}$, the getInfluenceObjects method returns a set (S_{ifl}) of all objects from the two sets (S_{cnd} and S_{prn}) which could influence $P(RNN_Q(B))$. With these objects, probabilistic pruning is performed. Depending on the result, the candidate is either discarded, added to the result set or verified. In the latter case, computation following [108] and [45] is performed to calculate the exact $P(RNN_Q(B))$. If this probability is above τ, the candidate can be confirmed as a result. In the following, we explain the individual modules in detail.

10.4.2 Spatial Pruning

The spatialPruning method (cf. Algorithm 9) performs a best-first search using a heap H prioritized by $minDist(Q, e)$, where e is an entry of the index I_{DB}. The heap is implemented such that the set of contained objects can be accessed without destroying the heap structure. For each de-heaped entry e, the predicate $DDC(e_2, e, Q)$ checks if this entry is pruned by another object or entry e_2 (contained in H, S_{prn} or S_{cnd}) according to Q, using a spatial domination decision criterion (in general DDC$_{OPT}$ is used here). If it

Algorithm 9 spatialPruning(Q, I_{DB}, S_{cnd}, S_{prn})

```
init min-heap $H$ with root entry of $I_{DB}$
while $H$ is not empty do
    de-heap an entry $e$ from $H$
    if $\exists e_2 \in H \cup S_{prn} \cup S_{cnd} : DDC(e_2, e, Q)$ then
        $S_{prn} = S_{prn} \cup \{e\}$
    else if $e$ is directory entry then
        for each child $ch$ in $e$ do
            insert $ch$ in $H$
        end for
    else if $e$ is data entry then
        $S_{cnd} = S_{cnd} \cup \{e\}$
    end if
end while
```

can be pruned, it is inserted in the S_{prn} set. Otherwise, if e contains a data object, it is added to the candidate set S_{cnd} and if e is a directory entry, its children are inserted into the heap.

The naive spatial pruning takes each pair of objects $A, B \in \mathcal{D}$ where $A \neq B$ and checks whether $Dom(A, B, R)$. Thus the runtime is $O(|\mathcal{D}|^2)$. In the average case, this step can be accelerated by the use of a spatial index structure to $(O(|\mathcal{D}| \cdot log(|\mathcal{D}|)))$, but the worst-case runtime remains $O(|\mathcal{D}|^2)$. The spatial pruning provides us with a set of candidate objects S_{cnd} where each candidate $C_i \in S_{cnd}$ is associated with a set of influence objects S^i_{ifl}.

Algorithm 10 getInfluenceObjects(Q, B, S_{cnd}, S_{prn})

```
$S_{ifl} = \emptyset$
for each $e \in S_{prn} \cup S_{cnd}$ do
    if $\neg(DDC(Q, e, B))$ then
        if $e$ is directory entry then
            $S_{prn} = S_{prn} \setminus \{e\}$
            for each child $ch$ in $e$ do
                $S_{prn} = S_{prn} \cup \{ch\}$
            end for
        else if $e$ is data entry then
            $S_{ifl} = S_{ifl} \cup \{e\}$
        end if
    end if
end for
return $S_{ifl}$
```

10.4.3 Probabilistic Pruning

For each candidate B, it is important for the next steps to find the objects which influence $P(RNN_Q(B))$. These are obviously the objects which might be closer to B than Q (objects e for which $P(e \prec_B Q \neq 0)$ This is done by Algorithm 10, which exploits Corollary

8.2 in order to exclude those objects that might not possibly prune B (for those objects $DDC(Q, e, B)$ holds). Each other data entry e for which $DDC(Q, e, B)$ does not hold is inserted into the S_{ifl} set.

The goal of the probabilistic pruning (cf Algorithm 11) is to estimate $P(RNN_Q(B))$ for a candidate B in a best possible way. If we can detect early that $P(RNN_Q(B)) > \tau$ or $P(RNN_Q(B)) < \tau$, further computation can be saved. The parameter *depth* is used to control how deep the hierarchical index structures (organizing the instances of each uncertain object) are resolved for more accurate pruning. Thus, the parameter offers to define a trade-off between accuracy and computational efficiency in the probabilistic pruning step.

Algorithm 11 probabilisticPruning(Q, B, S_{ifl}, τ, *depth*)

```
for 1...$depth$ do
  split($Q$)
  split($B$)
  $\forall I \in S_{ifl}$ split($I$)
  $P_{LB}(RNN_Q(B)) = 0, P_{UB}(RNN_Q(B)) = 0$
  for all $Q' \in Q$ and $B' \in B$ do
    $P_{LB}(RNN_{Q'}(B')) = 1, P_{UB}(RNN_{Q'}(B')) = 1$
    for each $I \in S_{ifl}$ do
      $P_{LB}(I \prec_{B'} Q') = 0$, $P_{UB}(I \prec_{B'} Q') = 1$
      for each $I' \in I$ do
        if $(I' \prec_{B'} Q')$ then
          $P_{LB}(I \prec_{B'} Q') = P_{LB}(I \prec'_B Q') + P(I')$
        else if $(Q' \prec_{B'} I')$ then
          $P_{UB}(I \prec_{B'} Q') = P_{UB}(I \prec_{B'} Q') - P(I')$
        end if
      end for
      $P_{UB}(RNN_{Q'}(B')) = P_{UB}(RNN_{Q'}(B')) \cdot (1.0 - P_{LB}(I \prec'_B Q'))$
      $P_{LB}(RNN_{Q'}(B')) = P_{LB}(RNN_{Q'}(B')) \cdot (1.0 - P_{UB}(I \prec'_B Q'))$
    end for
    $P_{LB}(RNN_Q(B)) = P_{LB}(RNN_Q(B) + P(Q')) \cdot P(B') \cdot P_{LB}(RNN_{Q'}(B'))$
    $P_{UB}(RNN_Q(B)) = P_{UB}(RNN_Q(B) + P(Q')) \cdot P(B') \cdot P_{UB}(RNN_{Q'}(B'))$
  end for
  if $P_{LB}(RNN_Q(B)) > \tau$ then
    return -1
  else if $P_{UB}(RNN_Q(B)) < \tau$ then
    return 1
  end if
end for
return 0
```

Let $S_{ifl} = \{I_1, ..., I_{|S_{ifl}|}\}$ denote the set of *influence objects* of B, which neither completely prune B w.r.t. Q nor are completely pruned by B. Only the set $\{I_1 \prec_Q B, ..., I_{|S_{ifl}|} \prec_Q B\}$ of random events has to be considered, since any object I_i for which $P(I_i \prec_Q B) = 0$ holds has no influence on $P(RNN_Q(B))$, and if there exists an object I_i for which it holds that $DDC(I_i, Q, B)$, then we can already conclude that $P(RNN_Q(B)) = 0$. But due to the problem of mutual dependencies between pruning events (cf. Section 8.4.2), here we cannot simply use the probability bounds $P_{LB}(I_i \prec\ _B Q)$ and $P_{UB}(I_i \prec\ _B Q)$ directly, as this would yield incorrect results. However, we can use the observation that the objects I_i are mutually independent and each candidate object I_i only appears in a single random variable $I_1 \prec_B Q, ..., I_{|S_{ifl}|} \prec_B Q$. Exploiting this observation, we can decompose[1] the objects $I_1, ..., I_{|S_{ifl}|}$ only to obtain mutually independent bounds for the probabilities $P(I_1 \prec_B Q), ..., P(I_{|S_{ifl}|} \prec_B Q)$, as stated by the following lemma:

Lemma 10.1. If B and Q are not decomposed, i.e. if $\underline{\mathcal{B}} = \{B\}$ and $\underline{\mathcal{Q}} = \{Q\}$, then $P(RNN_Q(B))$ is lower bounded by

$$P_{LB}(RNN_Q(B)) = \prod_{I_i \in S_{ifl}} 1 - P_{UB}(I_i \prec_B Q).$$

Proof. In a nutshell, the lemma can be proven by showing that the probability bounds $P_{UB}(I_i \prec_B Q), 1 \leq i \leq |S_{ifl}|$ can be computed independently of each other, since the computation of $DDC(I_i, Q, B)$ for $I_i \in \underline{\mathcal{I}}$ makes no assumptions on the positions of objects B and Q. This independence can be shown by exploiting that the bounds $P_{UB}(I_i \prec_B Q)$ are using the unpartitioned objects B and Q as parameters. Due to this independence, the probability bounds can simply be multiplied to derive the joint probability. □

An upper bound can be derived analogously:

$$P_{UB}(RNN_Q(B)) = \prod_{A_i \in I} 1 - P_{LB}(A_i \prec_Q B).$$

In summary, we can now derive, for each uncertain candidate object B a lower and an upper bound of the probability that B is an RNN of Q. However, these bounds may still be rather loose, since we only consider the full uncertainty region of B and Q so far, without any decomposition. Since the uncertain objects B and Q appear in each random event $I_i \prec_B Q$ $(I_i \in S_{ifl})$ that has to be evaluated, we cannot split the objects B and Q independently. Intuitively, the reason for this dependency is that any knowledge about the random event $I_i \prec_b Q$ may impose constraints on the position of B and Q. However, Lemma 10.1 directly yields the following corollary:

Corollary 10.1. Given partitions $B' \subseteq B$ and $Q' \subseteq Q$, then

$$P_{LB}(RNN_{Q'}(B')) = \prod_{I_i \in S_{ifl}} 1 - P_{UB}(A_i \prec_{Q'} B').$$

Note that the probability $P_{LB}(RNN_{Q'}(B'))$ is equal to the probability $P_{LB}(RNN_Q(B)|B \in B', Q \in Q')$, where $B \in B', Q \in Q'$ is a constraint to all possible worlds where B is located in partition B' and Q is located in partition Q'. This allows us to individually consider the subset of possible worlds where $B \in B'$ and $Q \in Q'$ and use Lemma 10.1 to efficiently compute $P_{LB}(RNN_{Q'}(B'))$ and $P_{UB}(RNN_{Q'}(B'))$. This can be performed for each pair

---

[1] e.g. using the bounding boxes of the R^*-trees of the objects

$(B', Q') \in \underline{\mathcal{B}} \times \underline{\mathcal{Q}}$, where $\underline{\mathcal{B}}$ and $\underline{\mathcal{Q}}$ denote the decompositions of B and Q, respectively. Now, we can treat pairs of partitions $(B', Q') \in \underline{\mathcal{B}} \times \underline{\mathcal{Q}}$ independently, since all pairs of partitions represent disjoint sets of possible worlds due to the assumption of a disjoint partitioning. Exploiting this independency, we can derive tighter bounds $P_{LB}(RNN_Q(B))$ and $P_{UB}(RNN_Q(B))$ for the probability that B is an RNN of Q by computing a lower and an upper bound of $P(RNN_{Q'}(B'))$ for each $(Q' \in \underline{\mathcal{Q}})$ and each $(B' \in \underline{\mathcal{B}})$ and then computing the weighted sum of these bounds as follows:

$$P_{LB}(RNN_Q(B)) = \sum_{B' \in \underline{\mathcal{B}}, Q' \in \underline{\mathcal{Q}}} P_{LB}(RNN_{Q'}(B')) \cdot P(B') \cdot P(Q'). \tag{10.1}$$

Thus in each iteration of the algorithm the involved objects (Q, B and all $I \in S_{ifl}$) are partitioned, which means we consider the partitioning at the i^{th} level of each local R*-tree. For each of these combinations, we first obtain bounds $P_{LB}(I \prec'_B Q')$ and $P_{UB}(I \prec'_B Q')$ of the probability that each $I \in S_{ifl}$ prunes B' w.r.t. Q', using Lemmas 9.1 and 9.2. Subsequently we multiply the complement of these bounds to acquire the lower (upper) bound probability $P_{LB}(RNN_{Q'}(B'))$ ($P_{UB}(RNN_{Q'}(B'))$) that no object $I \in S_{ifl}$ prunes B' according to Q'. Since all combinations $(Q' \in \underline{\mathcal{Q}}, B' \in \underline{\mathcal{B}})$ are mutually independent, the results can be summed up (weighting with the possible world probability of $(Q' \in \underline{\mathcal{Q}}, B' \in \underline{\mathcal{B}})$), to obtain the global probability bounds $P_{LB}(RNN_Q(B))$ and $P_{UB}(RNN_Q(B))$ according to Equation 10.1. This method returns -1 if the candidate B is PRNN of Q (with τ as threshold), and 1 if B can be pruned. If *depth* is not set to the maximum height of the R*-trees, it is possible that no decision can be made. In this case the method returns 0.

Regarding the runtime complexity the probabilistic pruning step considers each candidate object B S_{cnd} separately and partitions the candidate object, the query Q and the influence objects S^i_{ifl}. Assuming a branching factor of the spatial local index of b, each object consists of at most b^{depth} partitions, where *depth* is a parameter chosen before query processing. The domination relation $I' \prec_{B'} Q'$ is used for each pair of partitions $Q' \in Q, B' \in B$ and each partition I' of objects $I \in S_{ifl}$. This leads to a runtime of $O(b^{2 \cdot depth} \cdot |S_{ifl}| \cdot b^{depth}) = O(|S_{ifl}| \cdot b^{3 \cdot depth})$ for each candidate $B \in S_{cnd}$. In the worst case, where $|S_{ifl}| \in O(|\mathcal{D}|)$, $|S_{cnd}| \in O(|\mathcal{D}|)$ and $b^{depth} = m$ this yields a total runtime of $O(|\mathcal{D}|^2 \cdot m^3)$, where m is the number of instances in each object.

10.4.4 Verification

After the probabilistic pruning step, a smaller set of candidates $S'_{cnd} (\subseteq S_{cnd})$ remains. For each candidate $B \in S'_{cnd}$ verification is performed. Using the algorithm based on generating functions proposed in [108], this requires to sort all $m \cdot |S^i_{ifl}|$ instances of objects in S^i_{ifl} according to all m instances in B. We derive a runtime of $O(|S'_{cnd}| \cdot |S^i_{ifl}| \cdot m^2 \cdot log(|S^i_{ifl}| \cdot m))$. It can be observed, that for the case where m is large, the runtime of the probabilistic pruning step may exceed the runtime of the verification step. In our experimental section, we will verify this observation, and show how to choose values for the parameter *depth* such that this problem is avoided.

10.5 Probabilistic RkNN Queries

In this section we show how our proposed techniques can be extended to probabilistic RkNN queries. An RkNN query is defined as follows: Given a set of (certain) points P, a query object q, and a positive integer k, a reverse nearest neighbor query (RNN_q) returns all $p \in P$ which have q in their k-nearest neighbor set, formally ([152]): $RkNN_q = \{p \in P | dist(p,q) \leq dist(p, p_k)\}$, where p_k is the k-th nearest neighbor of p. In the context of uncertain objects, a $PRkNN_Q^\tau$ query returns the set of all objects $U_i \in \mathcal{D}$, for which the probability $P(U_i \in RkNN_Q)$ that U_i is a RkNN of Q is at least τ.

Approximation: Analogous to the $k = 1$ case, we approximate uncertain objects using *mbr*s and hierarchical partitioning.

Spatial Pruning: In the case where $k > 1$, the predicate $Dom(A, Q, B)$ must hold for at least k uncertain objects $A_i, 1 \leq i \leq |\mathcal{D}|$ in order to prune candidate B. Therefore, an uncertain candidate object can safely be pruned if there exist at least k objects for which the complete pruning criterion (cf. Lemma 7.1) holds. The complexity for the spatial pruning step is $O(|\mathcal{D}|^2)$ and independent of k, since in the worst case where a candidate B cannot be pruned, all objects may have to be considered.

Probabilistic Pruning: For probabilistic pruning, the task is to determine for a candidate object B, if its probability to be RkNN of Q is definitely less than τ (at least τ) in order to prune B (return B as a true hit). Analogous to the $k = 1$ case, we can derive the following bounds for the probability $P(A \prec_B Q)$ that $A \in \mathcal{D} \backslash B$ is closer to B than Q: a lower bound $P_{LB}(A \prec_B Q)$ and an upper bound $P_{UB}(A \prec_B Q)$. Given these bounds, we can apply the concept of uncertain generating functions (cf Section 9.3) in order to compute for each $0 \leq j < k$ a lower bound $P_{LB}(\#Pruners = j)$ and an upper bound $P_{UB}(\#Pruners = j)$ of the probability of the random event that for exactly j uncertain objects A_i, $A_i \prec_Q B$ is true. Bounds for the probability of the event $RkNN_Q(U_i)$ that U_i is a RkNN of Q can then be derived as follows:

$$P_{LB}(RkNN_Q(U_i)) = \sum_{0 \leq j < k} P_{LB}(\#Pruners = j)$$

$$P_{UB}(RkNN_Q(U_i)) = \sum_{0 \leq j < k} P_{UB}(\#Pruners = j)$$

An uncertain object U_i can be pruned if $P_{UB}(RkNN_Q(U_i)) < \tau$ and returned as a true hit if $P_{LB}(RkNN_Q(U_i)) > \tau$.

As shown in Chapter 9, the computational complexity is linear in k, yielding a total of $O(|DB|^2 \times k)$ for the probabilistic pruning.

Verification: The verification step can be performed analogously to the $k = 1$ case using the algorithm proposed in [32], which has been designed for $k \geq 1$. The total complexity of this algorithm is $O(|\mathcal{D}|^2 \cdot m^2 \cdot log(|\mathcal{D}| \cdot m) + k \cdot |\mathcal{D}|^2 \cdot m^2)$.

10.6 Related Work

To the best of our knowledge there exist the following two solutions for the PRNN problem.

10.6.1 LC Algorithm

Approximation: This algorithm is designed for the case where the appearance probability of uncertain objects is represented as a continuous PDF. Though it can easily be adapted to the discrete case. Each uncertain object is approximated by a sphere.
Spatial Pruning: The proposed pruning technique is based on trigonometric functions and can only be applied for spherical objects. Thus, it cannot be directly applied to the index pages (the authors use an R-tree as index structure). To overcome this shortcoming, each (rectangular) page of the index is at runtime approximated by a sphere containing this page.
Probabilistic Pruning: Additionally, a second sphere is computed for each database object in a preprocessing step. This sphere has the same center as the first sphere, but the radius is chosen as the minimal radius covering instances with a cumulated probability of at least $1 - \tau$. The idea of this approach is that if this second sphere can be pruned, then the corresponding object is pruned with a probability of at least $1 - \tau$, so it must have a probability less than τ to be an RNN of Q, and thus, it cannot be a PRNN of Q.
Verification: In the verification step, a range query around each candidate U_i is issued. The result contains all objects U_j such that $MinDist(U_j, U_i) < MaxDist(U_i, Q)$, i.e. all objects which affect $P(RNN_Q(U_i))$. Then $P(RNN_Q(U_i))$ is calculated by considering all possible worlds of the involved objects.

10.6.2 CLWZP Algorithm

Approximation: The CLWZP algorithm uses minimum bounding rectangles for the approximation of the uncertain objects. Additionally, each uncertain object has a local R-tree which organizes its instances.
Spatial Pruning: The pruning is performed using several pruning techniques arranged in series. The first used technique is MinMax. As shown in Section 3.2.2, MinMax is not complete, which means that, based on rectangular approximations, MinMax cannot detect valid pruning in all cases. Therefore, a second technique is proposed for special spatial relations of the query object and the pruner. If this technique cannot be applied, a general technique is used which considers all corners of the pruner for prune evaluation (see [45] for details). All proposed techniques (except MinMax) generate a pruning region defined by the pruner and the query. In this region objects can safely be pruned.
Probabilistic Pruning: Probabilistic pruning utilizes the generated pruning regions. Based on these regions, it may happen that only parts of a prunee get pruned. In this case, the prunee is trimmed down and further represented by an MBR containing all instances which could not be pruned (using a computational geometry algorithm). Additionally, the authors propose to partition object Q, to further improve the pruning.

Verification: In this phase, a range query is issued for each candidate U_i containing all objects affecting $P(RNN_Q(U_i))$. For each instance u_i of a candidate, the instances of these objects are sorted by the distance to u_i and inserted in a list. Based on these lists it is possible to calculate $P(RNN_Q(U_i))$.

10.6.3 Discussion

Although the LC algorithm is the only PRNN algorithm so far which can handle uncertain objects represented by a continuous PDF, it has the following drawbacks:

Parameter τ: Since the probabilistic pruning sphere has to be pre-computed using τ, it is not possible to change τ at query time. In a dynamic query environment however, the parameter may be adapted to the user's preferences, which is not possible in this approach.

Spherical Approximation: The main challenge, especially for the higher-dimensional case, is to find a small enclosing sphere of an uncertain object for effective pruning results. Finding the smallest enclosing sphere of an arbitrarily shaped object however has exponential runtime (w.r.t. to the number of vertices of the object), which allows only finding good but not best possible spheres in reasonable computational time. Additionally, as stated in [110], the spatial pruning technique is only conservative but not optimal for dimensions larger than 2. A third problem regarding the spherical approximation is the approximation of pages of the R-tree, which are rectangular by definition. A spherical approximation of an index page will therefore rarely be tight yielding low pruning power.

Verification: The verification step using integration of all remaining objects is based on the used uncertainty model. However, if the objects consist of discrete instances, there are more efficient solutions for the verification step (e.g. [108]). Note that also for the case where objects are represented by continuous PDFs, this step can be performed more efficiently, as we will show later.

The CLWZP algorithm however has a very complex spatial pruning technique (which is also used in the probabilistic pruning step) which requires 2^d distance calculations in the worst case (where d is the dimensionality of the data). This makes the approach practically inapplicable for the high-dimensional case. Just as the LC pruning, the CLWZP pruning is conservative which means that there exist cases where pruning is not performed although possible. Regarding the probabilistic pruning of CLWZP, the problem is that trimming requires expensive geometric computation but is used extensively in the algorithm.

10.7 Experimental Evaluation

In the experimental section, we compare our approach which we call HP (hierarchical pruning) with the two state-of-the-art PRNN query algorithms CLWZP [45] and LC [110]. For the implementation of CLWZP and LC we replaced R-trees by R*-trees wherever they were used. For the global R^*-tree we use a page size of 1024 byte. For the local R^*-trees indexing the instances of an uncertain object, we set the page size to a maximal capacity of three entries. The page size was chosen small in order to minimize CPU-cost, which we

parameter	**values synthetic**	**values real**
db size	**2000** - 10000	6216
dimensionality	2, **3**, 4, 5	2
# instances	50, **100**, 200, 400	100
τ	0.1 **0.2**, 0.3	**0.2**
maxdepth	0, 1, **2**, 3, 4	**2**
MBRextent	0.01, 0.02, 0.03, 0.04, **0.05**	N/A

Table 10.1: Parameters and their default values.

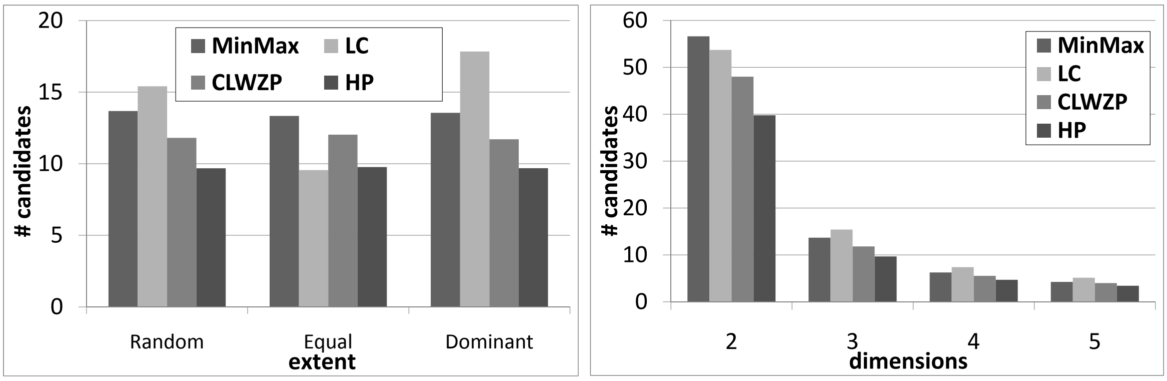
will see is the main bottleneck of a PRNN query. Whenever we had to access a local R^*-tree we treated this as a random access (Since the tree is very small the scanning process of reading the whole tree is negligible). We tested the three approaches under various parameter and data settings. For each setting, we performed 100 queries and averaged the measures. Since our experiments have been conducted against discrete uncertain datasets, we have adapted the LC algorithm to the discrete case for a fair comparison. The involved parameters and their default values (bold) can be seen in Table 10.1. For our experiments, we used one real-world dataset and several synthetic datasets to show the effect of changes in dimensionality and size. The datasets are described as follows: We generate synthetic uncertain objects in the $[0, 1]^d$ space by uniformly selecting the expected position of the objects in the space. A rectangle is generated around the expected position with a fixed total sum of side lengths (referred to as *extent*) with a default value of 0.05. By default, the extent is distributed uniformly on the dimensions, so there is a diversity of *mbr* shapes, some that are nearly cuboid, while others have a very large extent in few dimensions only. The object instances within the *mbr* are distributed uniformly by default.

As a real-world dataset, we utilize the International Ice Patrol (IIP) Iceberg Sightings Dataset[2]. This data set contains information about iceberg activity in the North Atlantic in the years 1960 to 2010. It contains the latitude and longitude values of 6216 sighted icebergs. An uncertain object is generated for each iceberg, by generating 100 Normal-distributed instances having a mean value corresponding to the position of its most recent sighting at the date 31.12.2009 and a variance corresponding to the time period between this date and the sighting. To avoid extreme impact of icebergs that have not been seen for decades, any instances outside a square of extent 0.0004 centered at the mean are cut off. The instances of an iceberg are Normal-distributed with a variance based on the time since the its sighting.

The experiments were run on a Windows 7 notebook with an Intel Core i5 processor (2.27 GHz), 6GB RAM.

---

[2] The IIP dataset is available at the National Snow and Ice Data Center (NSIDC) web site (*http://nsidc.org/data/g00807.html*).



(a) Varying extents. (b) Varying dimensions.

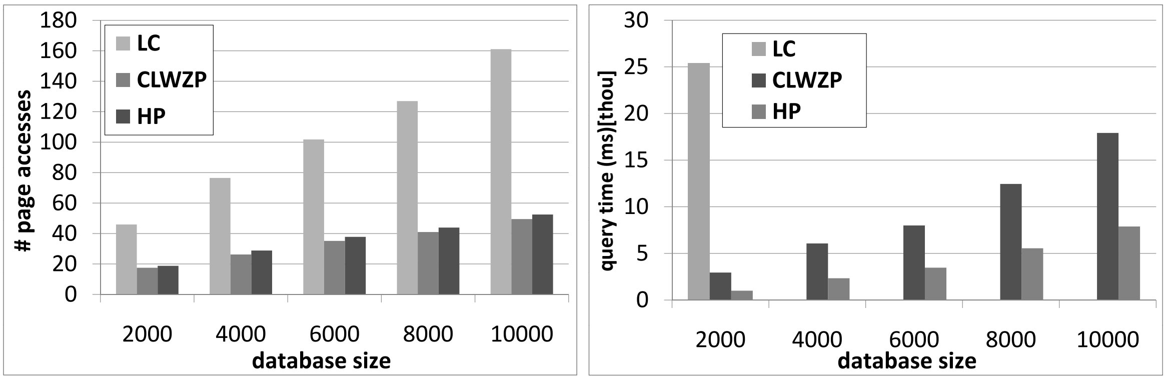
Figure 10.4: Comparison of different pruning techniques.

10.7.1 Spatial Pruning

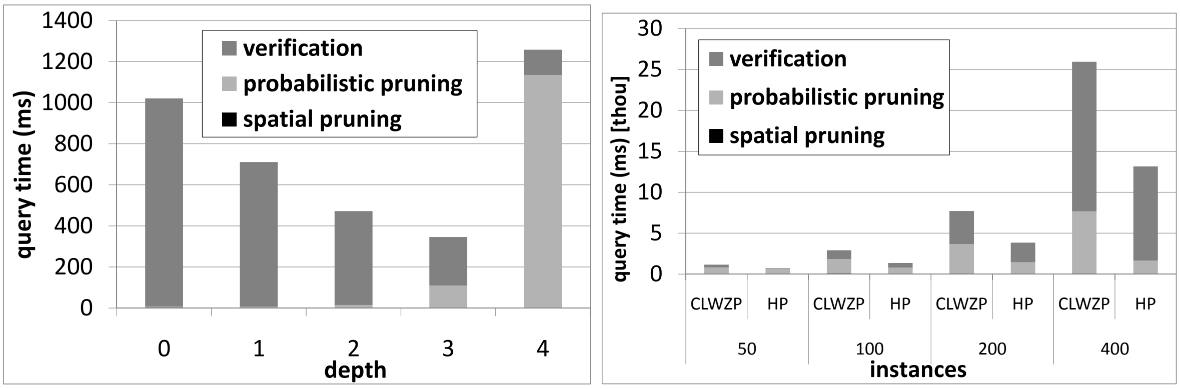
The first set of experiments compares the spatial pruning techniques of the three competing algorithms and additionally for completeness the MinDist/MaxDist based pruning (cf. Section 3.2.2) called *MinMax*. We generated 2000 uncertain 3-D objects uniformly distributed in the $[0, 1]^3$ space. Each object consists of 100 instances. 100 RNN queries were issued and we compared the number of candidates which were left after the spatial pruning step for the competing techniques. For the experiments shown in Figure 10.4(a), we tested the influence of different extensions in the dimensions of the objects. Three cases were compared: Random (random extension in each dimension, with fixed maximum), Equal (each dimension had the same extension = hypercube) and Dominant (one dimension has 5 times higher extension than the others). The results show that the spatial pruning technique from our algorithm has the highest pruning power in almost all settings. Only for objects equally extended in each dimension the LC-pruning is competitive. This is due to the reason that in this case, a sphere is a tight approximation for an uncertain object. However, in the other settings the LC-pruning yields the worst performance. We also compared the spatial pruning techniques for data with different dimensionality. Figure 10.4(b) illustrates that the advantage of HP-pruning is stable over varying dimensionality.

10.7.2 I/O-Cost

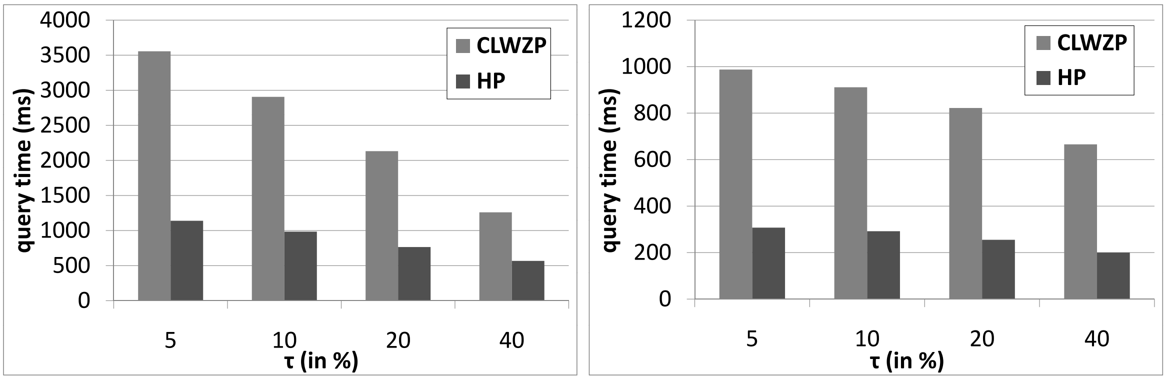
Next, we investigated the number of page accesses of the three algorithms needed on the global R*-tree (the index organizing the uncertain object approximations). The results are shown in Figure 10.5(a) for synthetic data with different database size. The LC algorithm needs by far the most page accesses which is reasonable, due to the handicaps mentioned in Appendix 10.6.3. The CLWZP algorithm performs even slightly better than the HP algorithm. The reason is mainly founded by the step to get the influence objects for each candidate. CLWZP here performs a search based on all instances of a candidate which produces a tighter bound than only using the approximation of the candidate (as



(a) Comparing page accesses. (b) Comparing cpu time.



(c) Influence of depth on HP. (d) Effect of sample size.



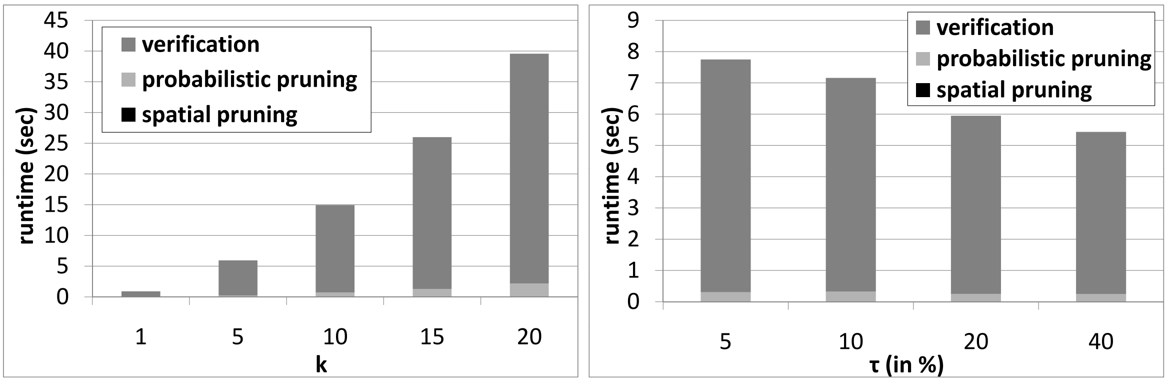
(e) Synthetic Data. (f) Real Data.

Figure 10.5: Performance of the PRNN Algorithm.

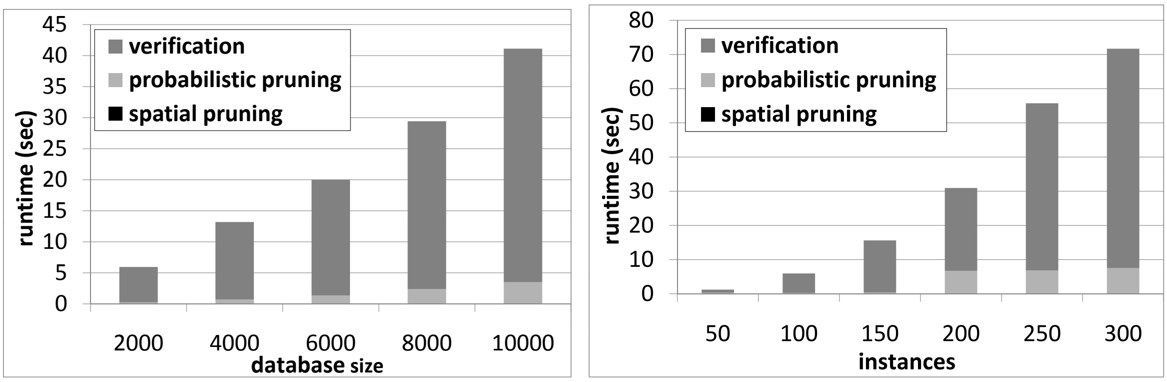
performed in HP). The drawback of this tighter bound is a much higher computational effort, which can be seen in the experiments in the next section. In summary, even for a small page size of 1 kB, the main bottleneck of a probabilistic RNN query are the CPU-cost, not the I/O-cost.

10.7.3 CPU-Cost

The Parameter *depth*: The main tuning parameter of the HP algorithm is *depth*. This parameter offers a tradeoff between the computational overhead for the probabilistic pruning and the verification step. This behavior can be seen in Figure 10.5(c), where by increasing *depth*, it is possible to dramatically reduce the CPU-cost in the verification step and thus the overall costs. It can be observed that for a low value of *depth*, the verification step is the main bottleneck. This is clear, since for a low value of *depth*, the set of partitions of each object X that is used for the probabilistic pruning is very small and contains large partitions. On the one hand, a small number of partitions leads to a very fast probabilistic pruning step, since only a small number of combinations of partitions has to be considered. On the other hand, the pruning power of the probabilistic pruning step is low in this case, due to the coarse approximations. The effect of the *depth* parameter in this setting is typical: there exists an optimal depth value $depth_{opt}$. Our experiments have shown that $depth_{opt}$ correlates to the height H of the R*-Tree. In all of our experiments, choosing $depth = H/2$ has shown to be a good heuristic. Determining better heuristics to find $depth_{opt}$ is part of our future work.
Scalability Experiments: Figure 10.5(b) shows the effects of the database size on the runtime of the LC, CLWZP and HP algorithms. It can be observed that the LC algorithm has an extremely high CPU-cost. The reason is that the LC algorithm was proposed for continuous uncertain objects and thus requires to consider the set of all possible worlds in the verification step, which is exponential in the number of influence objects. In contrast, both CLWZP and HP scale super-linearly in the database size. Regarding the effect of the number of instances size S, Figure 10.5(d) shows that the both algorithms CLWZP and HP also scale super-linearly. The reason is that both algorithms require to sort instances of a set of influence objects in the verification step, leading to a leading to a runtime of $O(S \cdot log(S))$.
Impact of τ: Figures 10.5(e) and 10.5(f) show that the runtime of the HP algorithm decreases for an increasing value of τ for both synthetic and real datasets. The rationale is that a high value of τ reduces the minimal probability $1 - \tau$ required for an uncertain object $A \in \mathcal{D} \setminus B$ to prune B.
PRkNN: We augmented our algorithm to answer PRkNN queries as proposed in Section 10.5 and evaluated the performance of this algorithm on the synthetic dataset using default parameters (cf. Table 10.1). In the first experiment (cf. Figure 10.6(a)), we varied the parameter k. It can be observed that the runtime scales slightly worse than linearly, which can be explained by the usage of uncertain generating functions that show a complexity of $O(k^2)$ ([30]). This is notable, since naive approaches need to consider all $\binom{N}{k}$ possible results. In the remaining experiments (Figures 10.6(b) to 10.6(d)) we evaluated the impact



(a) Dependency on k. (b) Effect of τ.



(c) Different database sizes. (d) Increasing sample size.

Figure 10.6: Behaviour of the PRkNN-Algorithm.

of the parameter τ, the database size and the number of instances per object (when setting $k = 5$). Each of these parameters scales equivalently to the $k = 1$ case. Note that in the experiments shown in Figure 10.6(d) we set $maxdepth = 3$ for more than 150 instances, matching our intuition of setting $maxdepth = H/2$.

10.8 Conclusions

In this chapter, we developed a general framework for probabilistic reverse nearest neighbor queries on uncertain data. We showed how two existing approaches and our new algorithm fit into the framework. Through new techniques to improve important parts of the framework, our algorithm is able to outperform the existing approaches under various settings. In addition, we proposed an efficient extension for probabilistic reverse k-nearest neighbor queries. An interesting starting point for future would be to evaluate techniques to dynamically adapt the parameter *depth* in the probabilistic pruning step to the distribution of instances within an object.

Part IV

Location Dependencies in Uncertain Spatio-Temporal Data

The only thing that makes life possible is permanent, intolerable uncertainty; not knowing what comes next.
Ursula K. LeGuin

Chapter 11

Uncertain Spatio-Temporal Data

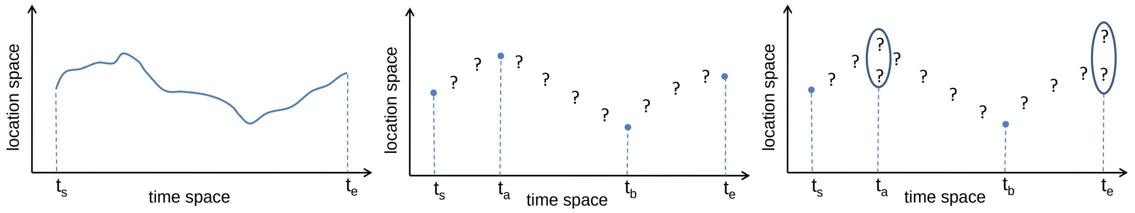
11.1 Introduction

With the wide availability of satellite, RFID, GPS, and sensor technologies, spatio-temporal data - data incorporating both location and time information - can be collected in a massive scale. Datasets containing such information are therefore becoming increasingly large, rich, complex, and ubiquitous. The efficient management of such data is of great interest in a plethora of application domains: from structural and environmental monitoring and weather forecasting, through disaster/rescue management and remediation, to Geographic Information Systems (GIS) and Tourist Information Systems. In most current research however, the assumption is made that the trajectory, i.e., the function of a spatio-temporal object that maps each point in time to a position in space, is known entirely without any uncertainty, such as the trajectory depicted in Figure 11.1(a). A survey on techniques for managing, querying and mining trajectory data without uncertainty is given in [180]. However, the physical limitations of the sensing devices or application dependent limitations of the data collection process introduce two sources of uncertainty:

- In real-life applications it is usually not possible to continuously capture the position of an object for each point in time. When recording the position of a car using GPS, the signal may for example be unavailable when the car passes a tunnel. In an indoor tracking environment where the movement of persons is captured using static RFID sensors, the position of the people in between two successive tracking events is also not available. Rather in such scenarios only a limited set of (location, time) pairs - so called *observations* - is available. In between these observations the exact values are missing (cf. Figure 11.1(b)).

- In most real-life applications the observed spatial location of an object at some point in time is not accurate. Using GPS or RFID for example the position of an object can only be determined with an accuracy of several meters. An other example is when the position of an object is inferred from different (partially) inconsistent sources. Thus the observations themselves may be imprecise or uncertain (cf. Figure 11.1(c)).

As an exemplary application, consider the problem of monitoring iceberg activity in the North Atlantic. Ships transiting between Europe and east coast ports of North America



(a) spatio-temporal data (b) missing observations (c) uncertain and missing observations

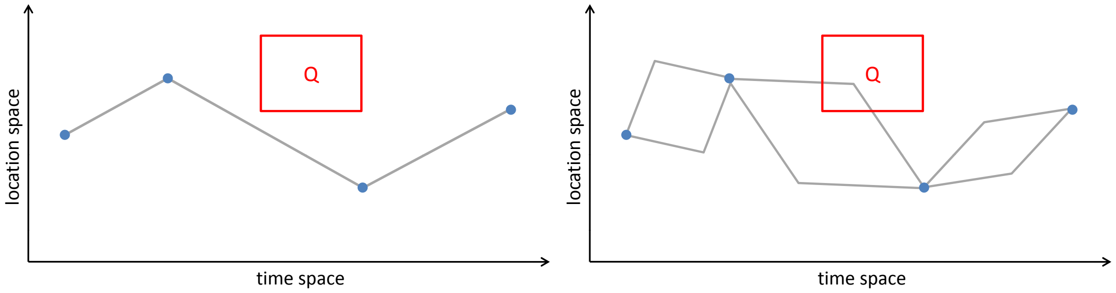
Figure 11.1: (Uncertain) Spatio-Temporal Data

traverse a great circular route that brings them into the vicinity of icebergs carried south by the cold Labrador Current. It was here that the R.M.S. Titanic sank in 1912, after it struck an iceberg. This disaster resulted in the loss of 1517 lives and led directly to the founding of the The International Ice Patrol (IIP) in 1914. The mission of the IIP is to monitor iceberg danger near the Grand Banks of Newfoundland and provide the locations of all known ice to the maritime community. The IIP does this by sighting icebergs, using visual observations from ships and aircrafts, as well as data from buoys and radars. A database stores the recorded positions and extents of observed icebergs and data models are used to predict their movement, based on the uncertainty of the recorded observations. Estimating the position of an iceberg at a time t underlies the two mentioned sources of error: (i) the obsoleteness of the most recent observation (cf Figure 11.1(b)) and (ii) the observation measurement error (cf Figure 11.1(c)). Using a model to describe this uncertainty, we can derive at each time t and each uncertain object o (i.e. an iceberg) a probability density function describing the position of object o at time t, and answer any related queries.

One of the most basic queries on spatio-temporal data (without uncertainty) is the following: *"find all objects within a given area (or at a given point) some time during a given time interval (or at a given time instant)"* [133]. This query is called *spatio-temporal window query*. Since we are dealing with uncertain data we have to adapt the definition of this query type. For the above example the query has to altered to *"find all objects within a given area (or at a given point) some time during a given time interval (or at a given time instant) with a sufficiently high probability"*. In fact we will formally define two different probabilistic spatio-temporal queries in Section 12.1 and for now rely on this informal introduction of the probabilistic spatio-temporal window query. Parts from this chapter have been published in [66] and [65].

11.2 Related Work

Uncertain data management has received a lot of attention in the past decade, due to the abundance of uncertain data, typically collected by the above mentioned monitoring devices. There exists a wide range of work studying spatial uncertainty in the static



(a) Linear interpolation (b) Spatio-temporal approximation

Figure 11.2: Models for UST data

(snapshot) case, where only one point of time is considered, (cf Section 8.3).

On the other hand, the problem of modeling and managing uncertain spatial data with a temporal component has surprisingly only received limited attention by the community. Existing works on uncertain spatio-temporal data management can be categorized into four groups: Models which remove the uncertain nature of the data by interpolation (similar to data cleaning), models which approximate the location of uncertain objects over time, models which treat time as an additional dimension to the uncertain location of the objects and models which use stochastic processes to model the possible movement of the uncertain objects.

11.2.1 Interpolation Models

In recent years, the research community often employed linear interpolation to model the movement of spatio-temporal objects [133, 137, 154], as depicted in Figure 11.2(a). [151] and [153] addressed the problem of query processing (window queries, joins, and nearest neighbour queries) under the assumption of linear interpolation. With the algorithms introduced in these works, the validity interval of a query result can be computed. However, clearly, the simple model of linearly interpolating between observations shows several drawbacks. On the one hand, whenever the sampling rate becomes low (observations are sparse), the true trajectory of an object might deviate greatly from the estimated trajectory. On the other hand it is even possible that a linearly interpolated trajectory becomes impossible: Imagine a car travelling from Munich to Berlin with sample points in exactly those two cities. The driver will never travel exactly along the straight line between the two cities because there are no streets following this trajectory. Although linear interpolation is one of the most common means of modelling the motion between observations in certain databases, other approaches have been evaluated as well. [150] introduced a framework that allows the future motion of objects to be described in a more complex manner than linear interpolation. Based on these complex motion functions, the authors aimed at predicting the future position of objects with minimized error. All approaches introduced so far do not take uncertainty into account. However, in scenarios where data are inherently uncertain, such as spatial and spatio-temporal databases, answering traditional queries using expected values and positions is inadequate, since the results could be incorrect [35]. This problem is also illustrated in Figure 11.2, where when using linear interpolation between the observations, the query is not satisfied (since the object does not pass the spatio-temporal window). However there might exist possible paths (worlds) between the observations where the object passes through the query window and thus satisfies the query. To mitigate this problem, several algorithms taking uncertainty into account have been developed.

11.2.2 Spatio-Temporal Approximation Models

The prevalent approach is to bound all possible (time, location) pairs of an object by a simple geometric structure in time and space. The result is a spatio-temporal approximation that is based on previous knowledge such as the maximum speed and the maximum acceleration of objects. An example for the one-dimensional case is shown in Figure 11.2(b), using the maximum speed of objects to obtain a conservative two-dimensional (space, time)-approximation. Generally, such approaches utilize various geometrical shapes, such as sheared cylinders [159, 160, 161], diamonds [121] and so called beads [101, 158] for the case of two spatial dimensions (the third dimension is time). Queries on these models include range queries [132, 160, 161] and kNN queries [159, 52]. The main problem of all these approaches is that no probability information is given for any object approximation. Thus, it is not possible to assess the probability of an object to satisfy some query predicate: For example, for the query window Q depicted in Figure 11.2(b), the only conclusion that can be made is that the approximated object may possibly intersect Q. However no information about the likelihood of this event can be assessed. In particular, this probability can be zero due to the conservative nature of these approximation models. Simple assumptions to estimate this probability, such as a uniform distribution over the conservative approximation, are often impractical: In practice, a vehicle having a fairly constant velocity between two (location, time) pairs is more likely than the same vehicle going at maximum speed, passing its destination, then performing a U-turn to race back the opposite direction in order to barely reach the second (location, time) pair in time.

11.2.3 Models treating time as a separate dimension

Some existing works treat spatio-temporal uncertainty as a simple extension of spatial uncertainty. For instance, the methods of [52] and [159] assume that for each timestamp in the history and every object, there is a location sample, which carries uncertainty. The samples are then modeled as uncertain regions, using probability density functions (PDFs) and these regions are connected to form the trajectories. Given a spatio-temporal range query [52], we can then estimate the probability that a trajectory intersects the query region by the overlap of the corresponding PDFs with the region. These types of models, however, may not be acceptable in the case where we only have a sample of locations in the moving history of an object. In this case, for timestamps where locations are not sampled, we have to infer the whereabouts of the object with the help of stochastic models. These models are particularly useful when predicting the future locations of objects, with the help of current (or recent) location and movement observations. In addition, these models allow for the economic representation and storage of the data, since only a subset of the object locations need to be sampled and used for the inference of their remaining locations.

In order to assess the actual probability of events in a spatio-temporal database, Mokhtar and Su [121] describe a model where the uncertainty region of each object is described by a time dependent stochastic process. Objects are given by MBRs which change their location and extent over time following the stochastic process. The paper shows how to answer certain types of window queries based on this model. However, describing the parameters of the uncertainty regions (instead of the object trajectories) by a stochastic process yields wrong results, i.e., results that cannot possibly be the correct result. The reason is that location dependency between consecutive timestamps is ignored by this model. Hence the position of an object at time t is assumed to be independent of its previous position at time t-1. A more detailed description of this basic problem is given in the next Section. Approaches like [15] and [172] consider uncertain time series (where the concrete value at each point of time is not known for certain) and data streams, respectively. Similar to other work, they disregard correlations between points in time.

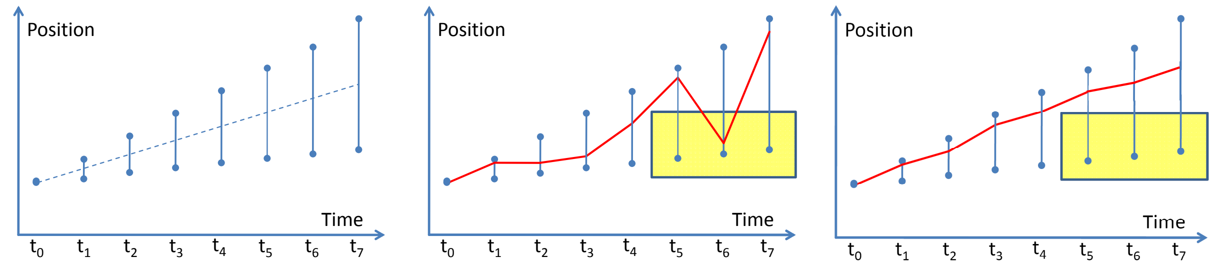
11.2.4 Models using stochastic processes

An initial approach to address the problem of location dependencies in UST data has been made in [135]. The authors assume a discrete state space and the Markov property to allow transitions between successive points of time. Their query language Lahar allows to formulate queries by stating regular expressions on an alphabet of states, and returns the probability of observing a sequence of states satisfying this regular expression. However, this syntax cannot handle time context as required by many common queries. For example, no regular expression can express the language that contains at least one character x at a given position interval. Such a query corresponds to a window query as used above.

In summary all existing approaches fail to answer probabilistic spatio-temporal (window) queries in a correct manner (according to possible worlds semantics). The approaches discussed in Section 11.2.1 and 11.2.2 are not able to provide any confidence for the result, the approaches reviewed in Section 11.2.3 disregard location dependencies which are crucial for window queries and the approach discussed in Section 11.2.4 is not powerful enough to answer probabilistic spatio temporal window queries.

11.3 Location Dependencies in Uncertain Spatio-Temporal Data

The approaches discussed in Section 11.2.3 suffer from the problem of treating the uncertain location of an object at each point of time independent from other timesteps. This proceeding yields a big problem, since it ignores *location dependencies.* To illustrate the



(a) Abstract Model (b) Assuming location independence (c) Not assuming location independence

Figure 11.3: location dependencies in UST data

problem of location dependencies, consider Figure 11.3(a), where an uncertain object trajectory is modeled. Here, it is assumed that an object moves forward in a one-dimensional space with an uncertain velocity. The velocity of o may change over time, but will not drop below some minimum speed greater than zero, and will not exceed some maximum speed (an example could be a sensor buoy on a river). Therefore, given the position of an object o at time t_0, the future position of the object can be modeled using the expected speed of o, the dashed line in Figure 11.3(a), as well as lower and upper bounds, or alternatively a variance, depicted by the intervals at each point of time. Under the assumption of temporal independence, at each point of time, the positions of o are modeled as independent random variables. Therefore, a trajectory as depicted in Figure 11.3(b) has a probability greater than zero. However, since o makes a large leap backward between times t_5 and t_6, this trajectory is not possible given the knowledge about the movement of o, which this model should incorporate. A possible trajectory is shown in Figure 11.3(c). Here, the object moves within its speed limits at each point of time. The flaw of modelling trajectories which are not actually possible becomes a problem when processing spatio-temporal queries based on this model. For example, consider a spatio-temporal window query, which returns for an object o the probability that o intersects the query window q, depicted in Figure 11.3(b) and Figure 11.3(c). For any model that ignores the dependency between locations at subsequent points of time, the probability that o is always outside of the window is the product of many probabilities, thus becoming very small. Therefore, for a large number of points of time inside the query region, the probability that o intersects the query window converges to one. However, if the dependency between locations at subsequent points of time is considered, then the probability that o is outside of the window at time t_6 depends on the probability at time t_5 in this example: If o is not in the window q at t_5, then it cannot be in q at t_6 either, since the object cannot move backwards. Thus, the probability that o intersects the query window q at any time, is equal to the probability that o intersects the query window at time t_5. Thus, an important aim of adequate models for UST data is to properly model such location dependencies, instead of simply treating time as an additional dimension in space.

The rest of this part is organized as follows: In Chapter 12 we will develop a basic model based on Markov Chains to describe uncertain spatio-temporal data. Based on this model we will show how to answer different types of probabilistic spatio-temporal window queries according to possible worlds semantics. Chapter 13 then discusses an indexing technique for UST data, which is able to speed up scan-based query processing by several orders of magnitudes. Finally Chapter 14 considers other types of (similarity) queries on UST data and offers a general framework for efficiently approximating query results based on a efficient sampling technique.

Chapter 12

Querying Uncertain Spatio-Temporal Data

In this chapter we present a framework for efficiently modeling and querying uncertain spatio-temporal data. The key idea of our approach is to model possible object trajectories by stochastic processes. This approach has three major advantages over previous work. First it allows answering queries in accordance with the possible worlds model. Second, dependencies between object locations at consecutive points (as discussed in Section 11.3) in time are taken into account. And third it is possible to reduce all queries on this model to simple matrix multiplications. Based on these concepts we propose efficient solutions for different probabilistic spatio-temporal queries for a particular stochastic process, the Markov Chain. In an experimental evaluation we show that our approaches are several orders of magnitudes faster than state-of-the-art competitors. Parts from this chapter have been published in [66].

The rest of this Chapter is organized as follows: In Section 12.1 we formally describe the problem of modelling UST data and define important query types on this data. Subsequently in Section 12.2 we illustrate how UST data can be modelled using Markov Chains. Section 12.3 presents two methods for query processing on the Markov Chain model. Afterwards the techniques are extended in order to handle multiple observations in Section 12.4 and two additional query types are discussed in Section 12.5. Sections 12.6 and 12.7 evaluate the proposed techniques and conclude the chapter, respectively.

12.1 Problem Definition

In this part, we assume discrete space and time domain $\mathcal{S}$ and $\mathcal{T}$, where space is defined by coordinates and time denotes points in time. Formally, let $\mathcal{S} = \{s_1, ...s_{|\mathcal{S}|}\} \subseteq \mathbb{R}^d$ be a finite set of possible locations in space which we call *states* and let $\mathcal{T} = \mathbb{N}_0^+$ be the time-space. Consequently, a (certain) object o that moves in space is represented by a *trajectory* given by a function $o : \mathcal{T} \rightarrow \mathcal{S}$ that defines the location $o(t) \in \mathcal{S}$ of o at a point of time $t \in \mathcal{T}$.

We assume uncertain spatio-temporal objects, i.e. objects $o \in \mathcal{D}$ that are associated

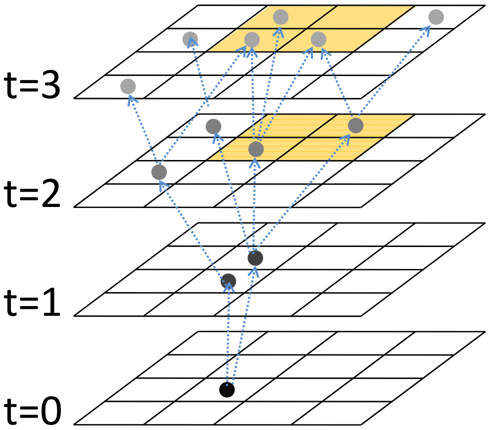


Figure 12.1: Querying Uncertain Spatio-Temporal Data

with *uncertain object trajectories*. We have to cope with uncertain trajectories when taking future motions of objects (e.g., by extrapolation) into account or if we want to infer locations between subsequent object observations (i.e., by interpolation). In some applications, like the iceberg tracking example in the introduction (cf Section 11.1), observations are rare and we have to infer the object locations at most timestamps in the domain $\mathcal{T}$. An observation at a specific time may be precise or uncertain. To model uncertain object trajectories, we suppose that the locations of an uncertain spatio-temporal object $o \in \mathcal{D}$ at time t are realizations of a random variable $o(t)$. An uncertain object trajectory of object $o \in \mathcal{D}$ comprises a set of trajectories, each assigned with a probability indicating its likelihood to be the true trajectory of o. This consideration suggests modeling uncertain object trajectories as a realization of a stochastic process [92], formally:

Definition 12.1 (Uncertain Object Trajectory)**.** Given the spatial domain $\mathcal{S}$ and the time domain $\mathcal{T}$, an *uncertain object trajectory* $o(t) \in \mathcal{S}$ of an object $o \in \mathcal{D}$ is a stochastic process $\{o(t) \in \mathcal{S}, t \in \mathcal{T}\}$.

An example of an uncertain object trajectory of an object $o \in \mathcal{D}$ is illustrated in Figure 12.1. The raster models all possible locations (i.e., states) in $\mathcal{S}$, shown for the time sequence $t = \langle 1, 2, 3, 4 \rangle$. Here we assume that the last observed location of object o was at time $t = 0$; all locations of o that follow are uncertain. Consequently, the uncertain trajectory of o comprises all possible trajectories starting at $t = 0$.

Our goal is to efficiently evaluate probabilistic spatio-temporal queries on uncertain spatio-temporal objects; i.e., queries about objects that are probably located in a given spatial region during a given range in time. We assume a set of uncertain spatio-temporal objects $\mathcal{D}$, i.e. objects associated with uncertain object trajectories $o(t)$, and focus on spatio-temporal queries specified by the following parameters: (i) a spatial region $S^{\Box} \subseteq \mathcal{S}$, i.e. a set of (not necessarily connected) locations in space, and (ii) a set $T^{\Box} \subseteq \mathcal{T}$ of (not necessarily subsequent) points in time. In the remainder, we use $Q^{\Box} = S^{\Box} \times T^{\Box}$ to denote the query ranges in the space and time domain. The most intuitive definition of a probabilistic spatio-temporal query is given below:

Definition 12.2. [Probabilistic Spatio-Temporal (Exists) Query] Given a query region $S^\square$ in space and a query region $T^\square$ in time, a *probabilistic spatio-temporal exists query* (PST∃Q), retrieves for each object $o \in \mathcal{D}$ the probability $P^\exists(o, S^\square, T^\square)$ that o is located in $S^\square$ at some time $t \in T^\square$.

This query type has been studied before (e.g. in [161, 160]), albeit over data models that disregard dependencies between locations at consecutive timestamps, as we have discussed in Section 11.2. For our motivating application described in Section 11.1, an exemplary query could be: find all icebergs that have non-zero probability to be inside the movement range of a particular ship during the ship's movement in the North Atlantic. Another query could be to predict the number of cars that will be in a congested road segment after 10-15 minutes. In addition, we study the following two interesting probabilistic query variants. Note that the second variant has not been considered in the past:

Definition 12.3. [Probabilistic Spatio-Temporal For-All Query] A *probabilistic spatio-temporal for-all query* (PST∀Q) retrieves for each object $o \in \mathcal{D}$ the probability $P^\forall(o, S^\square, T^\square)$ that o remains in $S^\square$ for *all* times $t \in T^\square$.

Definition 12.4. [Probabilistic Spatio-Temporal k-Times Query] A *probabilistic spatio-temporal k-times query* (PSTkQ) retrieves for each object $o \in \mathcal{D}$ and each parameter $1 \leq k \leq |T^\square|$ the probability $P^{k-times}(o, S^\square, T^\square)$ that o is located in $S^\square$ at *exactly* k times $t \in T^\square$.

PST∀Q and PSTkQ are important complements to the PST∃Q. For example, these queries can progressively determine candidates that remain in a certain region for a while. For example, for a given region somewhere in the north Atlantic we want to retrieve all icebergs that have non-zero probability remaining in this region for a specified period of time, e.g. to be able to make some measurements over a certain time period. Further examples where such queries are useful are for location-based-service (LBS) applications, e.g. a service provider could be interested in customers that remain in a certain region for a while, such that they can receive advertisements relevant to the location.

Note that, although the spatial (temporal) parameters of the queries define contiguous regions (intervals) in the space (time) domain, our query processing approaches are also applicable for any arbitrary subset of the space (time) domain.

12.2 Modeling Uncertain Spatio-Temporal Data

In accordance to the previous section, the constituent parts of an uncertain spatio-temporal object are specifications of probability distributions over the space and time domain. Naive models typically restrict the regions that the object can be at each timestamp. All uncertain trajectories are combinations of locations in these regions, giving all these trajectories equal probabilities ([159, 161, 160, 15, 172, 121]). However, as already mentioned, these models disregard any location dependencies between successive points in time, possibly yielding incorrect results.

According to Definition 12.1, the uncertain motion of an object is defined as a stochastic process. In particular, the (first-order) Markov-Chain model in a discrete time- and state-space is assumed. The state space of the model is the spatial domain $\mathcal{S}$. State transitions are defined over the time domain $\mathcal{T}$. In addition, the Markov-Chain model is based on the assumption that the position $o(t+1)$ of an uncertain object o at time $t+1$ only depends on the position $o(t)$ of o at time t. Formally:

Definition 12.5. A stochastic process $o(t), t \in \mathcal{T}$ is called a Markov-Chain if and only if

$$\forall t \in \mathbb{N}_0 \forall s_j, s_i, s_{t-1}, ...s_0 \in S :$$

$$P(o(t+1) = s_j | o(t) = s_i, o(t-1) = s_{t-1}, ..., o(0) = s_0) =$$
$$P(o(t+1) = s_j | o(t) = s_i)$$

The conditional probability

$$M^o_{i,j}(t) := P(o(t+1) = s_j | o(t) = s_i)$$

is the (single-step) *transition probability* of o from state s_i to state s_j at a given time t. Transition probabilities are represented by a matrix $M^o(t)$, called *transition matrix of object o at time t.*

Let $P(o,t)$ be the distribution vector of an object o at time t, such that $P_i(o,t) = P(o(t) = s_i)$ corresponds to the probability that o is located at state s_i at time t. The distribution vector of o at time $t+1$ can be inferred from $P(o,t)$ as follows:

Corollary 12.1.

$$P(o, t+1) = P(o,t) \cdot M^o(t)$$

The (single-step) transition matrix $M^o(t) \in \mathbb{R}^{S^2}$ is a *stochastic* matrix, i.e. the following properties hold:

$$\forall i, j \in S : M_{i,j} \geq 0$$
$$\forall i \in S, \sum_{j \in S} M_{i,j} = 1$$

Definition 12.6. A Markov Chain is homogeneous if and only if the transition probabilities are independent of t, i.e. $M_{i,j}(t) = M_{i,j}$.

An instantiation of the object at each point of time is called a *possible world* (or possible trajectory). Figure 12.2 depicts a small subset of all possible worlds of an uncertain object for a given Markov-Chain model.

Note that in this work we assume that the transition probabilities $M^o_{i,j}(t)$ are given, e.g. derived from expert knowledge or derived from historical data. For example, the current of the water in the Atlantic ocean can be used to infer the transitions of icebergs. For traffic data, the transition probabilities at each road intersection are usually estimated by historic traffic records. This approach models the movement between successive points

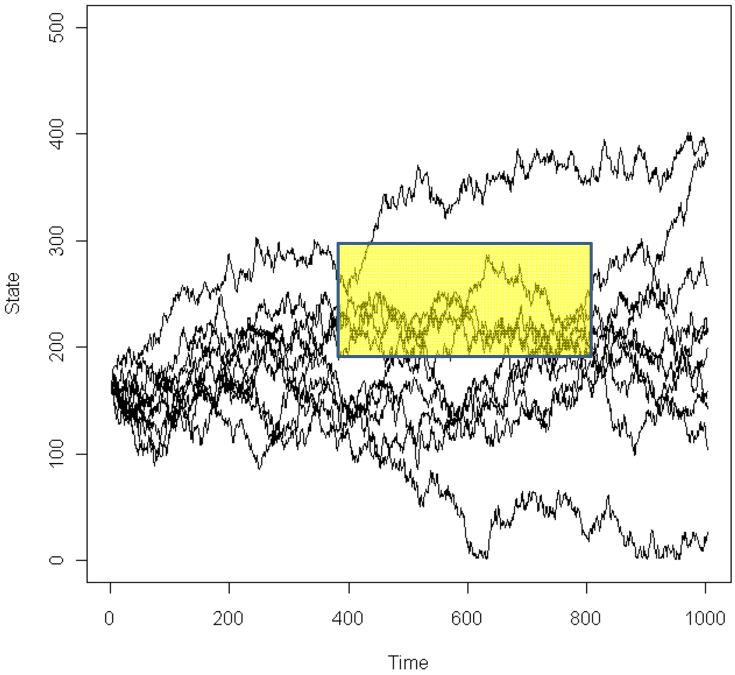


Figure 12.2: Some possible worlds of one uncertain object

in time, based on background knowledge (e.g., physical laws) and has proven capable of effectively capturing the behavior of real objects in practice. For instance, [12] and [79] show how Markov-models can effectively capture the movement of vehicles on road networks for prediction purpose. In [135], it is shown that Markov-models can also be used to model the indoor movement of people, as tracked in RFID applications. As shown in [118], Markov Chains can be employed for modelling human movement.

For simplicity we will model any uncertain object $o \in \mathcal{D}$ by a homogeneous Markov-Chain and we assume the same model is valid for all objects in the database. Let us note that the proposed methods are also applicable for the general case of inhomogeneous and individual Markov Chains (with the query-based processing technique from Section 12.3.2 being an exception since it requires all objects in the database to follow the same model).

The main challenge in answering probabilistic spatio-temporal queries is to correctly consider the possible world semantics in the model. In other words, the issue at hand is to return the fraction of possible worlds of an object o, for which it holds that at any time $t \in T^{\square}$ it holds that $o(t) \in S^{\square}$. Again considering Figure 12.2, any world, for which the corresponding path intersects the spatio-temporal query window, satisfies the query predicate (for a PST∃Q).

Still, the number of possible worlds is in $O(|S|^T)$; i.e., exhaustively examining all of them requires exponential time, even for finite time and space domains. Clearly, any naive approach that enumerates all possible worlds, is not feasible. An overview of the most important notations used in this part of the thesis is given in Table 12.1.

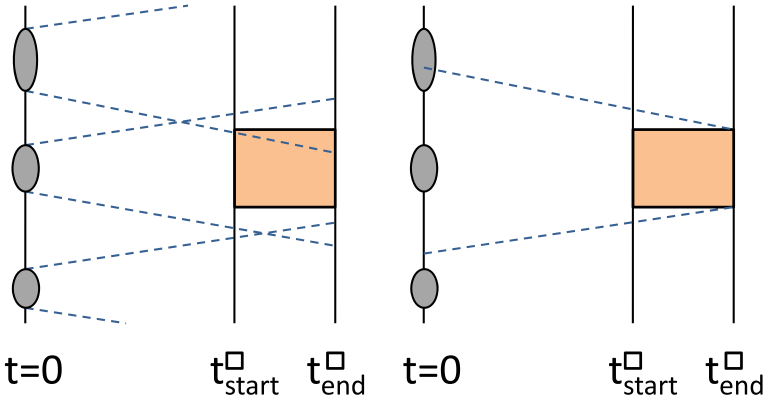
Symbol	**Meaning**
$o \in DB$	a UST object from the database
$o(t)$	random variable describing the location of o at time t
$\mathcal{S} = \{s_1, \ldots, s_{\mathcal{S}}\}$	the state space
$\mathcal{T} = \{t_1, \ldots, t_{\mathcal{T}}\} \in \mathbb{N}$	the time space
$M^o(t)$	transition matrix of object o at time t
$M^o_{ij}(t)$	$= P(o(t+1) = s_j \mid o(t) = s_i)$
$P(o,t)$	distribution vector of o at time t
$P_i(o,t)$	$P(o(t) = s_i)$
$\Theta^o = \{\Theta^o_1, \ldots, \Theta^o_{\lvert\Theta^o\rvert}\}$	set of observations for object o
$\Theta^o_i =< t^o_i, \theta^o_i >$	an observation ((time, location)-pair) of object o
θ^o_i	distribution vector over the state space, in case of a certain observation θ^o_i may also represent the state s_{ob} where the observation took place

Table 12.1: Table of notations

12.3 Probabilistic Spatio-Temporal Query Processing using the Markov-Chain Model

In this section, we show how the queries that we presented in Section 12.1 can be evaluated efficiently. For the ease of presentation, we first assume that for each object, there is a single observation at time $t = 0$ and that we want to predict the result of a probabilistic spatio-temporal exists query (Definition 12.2) in the future. We generalize our results for the case of multiple observations and other queries in Sections 12.4 and 12.5, respectively.

We propose two approaches towards efficient query processing. The *object-based approach* (Section 12.3.1) directly computes $P^{\exists}(o, S^{\Box}, T^{\Box})$ in an efficient way, while the *query-based approach* (Section 12.3.2) follows a reverse methodology: computation starts at the query window and the transposed Markov-Chain matrix is used to compute, for each state $s \in \mathcal{S}$ the probability that an object starting at s satisfies the query predicate. Figure 12.3 shows a high-level example of query evaluation using these two approaches. Consider a spatio-temporal query with query time interval $[t^{\Box}_{start}, t^{\Box}_{end}]$ illustrated by the shaded rectangle on the right of each subfigure. We would like to predict the result of the query, based on observations about the locations of the objects at time ($t = 0$) and a given model that captures their state transition probabilities (states correspond to spatial locations illustrated by the y-axis in the example). Exemplary objects are shown in oval shapes. The object-based approach, illustrated in Figure 12.3(a), examines for each object the possible trajectories (i.e., possible worlds) that the object will follow in the future and finds the probabilities of trajectories that intersect the query window. For instance, the top-most object has a non-zero probability to be a result of the query, the middle object has a higher probability and the object at the bottom has a zero probability. To compute the prob-



(a) Object based approach (b) Query based approach

Figure 12.3: Schematic illustration of object-based (OB) and query-based (QB) query processing.

abilities of all possible worlds that intersect the window, for each object, we use matrix multiplications. We show how pruning techniques can be incorporated into the matrix multiplications for efficiency. Still, this approach iteratively examines all objects in the database exhaustively. The query-based approach (Figure 12.3(b)), on the other hand, *reverses* the computation: given that an object intersects the query window, we compute the probability that this object corresponds to any of the objects in the database; this way, examining objects that are irrelevant to the query is avoided, while at the same time the query results are computed in batch. We now describe these two approaches in detail.

12.3.1 Object-Based Query Processing

Given an object o, in order to find the probability that o is part of a query result, we could use the following straightforward approach: compute, for each point of time $t \in T^{\square}$ the probability that o is located in $S^{\square}$ and aggregate these probabilities. Clearly, this approach yields incorrect results (cf Section 11.3). The problem of this approach is that a specific possible world (i.e., trajectory) may be considered more than once, if it overlaps with the query at multiple timestamps. Therefore, to correctly process queries, possible worlds which satisfy the query predicate should only be considered once in the computation of the result.

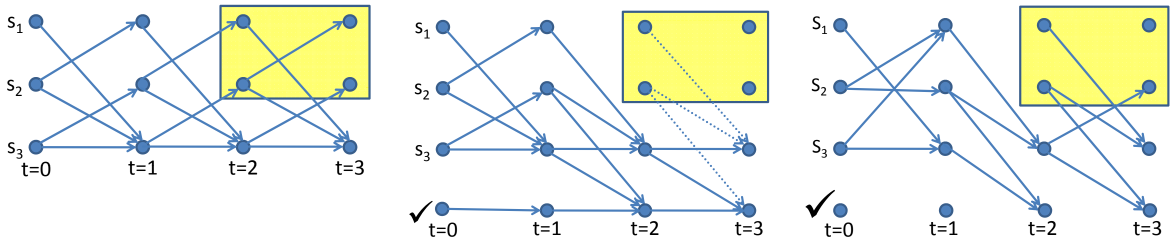
As an example of proper query evaluation, consider the following Markov-Chain with three states:

$$\begin{pmatrix} 0 & 0 & 1 \\ 0.6 & 0 & 0.4 \\ 0 & 0.8 & 0.2 \end{pmatrix},$$

for which all possible worlds are depicted in Figure 12.4(a) for the first four points of time.[1]

---

[1] Probabilities are omitted for readability, but can be found in the Markov-Chain.



(a) All possible worlds (b) Object-based approach (c) Query-based approach

Figure 12.4: Procedure of object-based (OB) and query-based (QB) query processing.

Additionally, assume a window query defined by $S^\square = \{s_1, s_2\}$ and $T^\square = \{2, 3\}$ and assume an object o which has been observed at s_2 at time $t = 0$, i.e. $P(o, 0) = (0, 1, 0)$. To compute the probability $P^\exists(o, S^\square, T^\square)$ that o intersects the query window, we first compute the probability distribution $P(o, 2)$ at time $t = 2$, using Corollary 12.1 two times. The resulting probability vector $P(o, 2) = (0, 0.32, 0.68)$ gives us a lower bound of 32% for $P^\exists(o, S^\square, T^\square)$, since any world in which o is located at state s_2 at time $t = 2$ satisfies the query window. Thus, these worlds can already be considered as true hits and must be ignored at future points of time. Thus, we obtain a new probability distribution $P(o, 2)' = (0, 0, 0.68)$ which will be used for the next state transition using Corollary 12.1. The resulting vector $P(o, 3) = (0, 0.544, 0.136)$ means that out of the remaining 68% of worlds which have not already been reported as true hits, another 54.4% can now be returned as true hits, since in these worlds o is located at s_2 at time $t = 3$. Since $t = 3$ is the last point of time belonging to the query window, the fraction of 0.136 worlds can be reported as true drops, since in these worlds, o has not intersected the query window at any $t \in T^\square$. Thus, the result of this query is $0.32 + 0.544 = 0.864$.

The above example gives an intuition on how to answer queries correctly by identifying worlds that satisfy the query and excluding them from further processing. We incorporate this technique directly in the Markov-Chain, to facilitate efficient on-the-fly pruning when matrix multiplication is performed between a state vector and the Markov-Chain. To achieve this goal, we introduce a new state which is denoted by "✓", denoting true hits, i.e. for any world in which an object o reaches ✓, we know that o satisfies the query predicate. This ✓ state satisfies the *absorbing* property, i.e. any world reaching this state cannot leave it. Instead of directly using the Markov-Chain M, we build the following two new matrices

$$M^- = \begin{pmatrix} M & zero(|S|) \\ zero(|S|)^T & 1 \end{pmatrix} \text{ and } M^+ = \begin{pmatrix} M' & sum(S^\square) \\ zero(|S|) & 1 \end{pmatrix},$$

where $zero(|S|)$ is a vector of size S containing zeroes only, $zero(|S|)^T$ is its transposed, M' is derived from M by replacing all columns that correspond to states in $S^\square$ by zero vectors, and $sum(S^\square)$ is a column vector containing for each line in M the sum of values removed this way. The idea is to use M^- outside of the query window time and M^+ during the time when the query window is active. By using the transition matrix M^+ all possible worlds which satisfy the query are pushed into the absorbing state.

The initial object distribution vector of an object o is now extended by an additional value of zero, corresponding to the fact that initially (at time $t = 0$) we cannot identify any worlds which satisfy the query predicate.[2] At each state transition, where the target state does not belong to $T^\Box$, we can now use M^- instead of M, which has the same effect as M, while preserving the probability of state $\checkmark$. If the target state belongs to $S^\Box$, then M^+ is used instead. This way, worlds leading into states in $S^\Box$ are now redirected to state $\checkmark$ instead.

Example 12.3.1. For the matrix $\begin{pmatrix} 0 & 0 & 1 \\ 0.6 & 0 & 0.4 \\ 0 & 0.8 & 0.2 \end{pmatrix}$ of our running example, the corresponding new matrices are $M^- = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0.6 & 0 & 0.4 & 0 \\ 0 & 0.8 & 0.2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ and $M^+ = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0.4 & 0.6 \\ 0 & 0 & 0.2 & 0.8 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

The corresponding visualization is depicted in Figure 12.4(b). Here M^- is used for the first transition from $t = 0$ to $t = 1$, while for the transitions from $t = 1$ to $t = 2$ and from $t = 2$ to $t = 3$, M^+ is utilized as transition matrix. Thus, for an object that has been observed at state s_2 at time $t = 0$, we obtain the initial probability distribution vector $P(o, 0) = (0, 1, 0, 0)$, where the fourth value denotes the initial probability of being a true hit, which is zero since $t = 0$ does not belong to $T^\Box$. The transition to $t = 1$ yields $P(o, 1) = P(o, 0) \cdot M^- = (0.6, 0, 0.4, 0)$. Clearly, the fourth value corresponding to state $\checkmark$ is zero, since no state $t \in T^\Box$ has been visited so far. Transition to $t = 2$ yields $P(o, 2) = P(o, 1) \cdot M^+ = (0, 0, 0.64, 0.36)$ and the final transition yields $P(o, 3) = P(o, 2) \cdot M^+ = (0, 0, 0.136, 0.864)$. Therefore, the resulting probability that o intersects the query window is 0.864.

12.3.2 Query-Based Query Processing

The object-based approach applies for each object o the methodology described in Section 12.3.1 to compute the probability that o is part of the query result. The query-based approach assumes that an object intersects the query. Based on this assumption, this approach starts at the last point of time in the query window, and goes backward in time using the transposed Markov-Chain. This way, we can compute, at any time t, the probability vector $P(t)$ containing for each state s, the probability that an object starting at state s at time t will satisfy the query predicate.

Therefore, we start at time $t^\Box_{end} := max(T^\Box)$. Clearly, at $t^\Box_{end}$, a path satisfies the query predicate if and only if it is in state $\checkmark$: If it is not in state $\checkmark$ at time $t^\Box_{end}$, then it will

---

[2] In the special case where $t = 0$ belongs to $T^\Box$, we adjust the initial vector by moving all probabilities of states in $S^\Box$ to state $\checkmark$.

never reach state ✓, since at any time after $t^{\square}_{end}$, the matrix M^- will be used which does not allow to enter state ✓. Thus, the vector $P(t^{\square}_{end})$ has the form $(0, ..., 0, 1)$. Intuitively, this vector corresponds to the assumption, that a path satisfies the query. Now we go back in time using this assumption by using the transposed matrices $(M^-)^T$ and $(M^+)^T$: If the current state belongs to $S^{\square}$, we use $(M^+)^T$, otherwise, we use $(M^-)^T$. This procedure is repeated until $t = 0$, the last point of time at which an object o has been observed, is reached. The resulting vector v yields, for each state $s \in \mathcal{S}$, the probability that an object starting at s (with a probability of one) satisfies the query. Vector multiplication of the probability distribution $P(o, 0)$ with v yields the result probability $P^{\exists}(o, S^{\square}, T^{\square})$.

Example 12.3.2. Again, consider the example depicted in Figure 12.4(c), where we want to compute the probability that an object o intersects the query $S^{\square} = \{s_1, s_2\}, T^{\square} = \{2, 3\}$. By transposing M^- and M^+, we obtain:

$$(M^-)^T = \begin{pmatrix} 0 & 0.6 & 0 & 0 \\ 0 & 0 & 0.8 & 0 \\ 1 & 0.4 & 0.2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \text{ and } (M^+)^T = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0.0 & 0 \\ 1 & 0.4 & 0.2 & 0 \\ 0 & 0.6 & 0.8 & 1 \end{pmatrix}$$

The query-based approach starts by assuming the probability distribution $P(t = 3) = (0, 0, 0, 1)$. Since $t = 3 \in T^{\square}$, we compute

$$P(t = 2) = P(t = 3) \cdot (M^+)^T = (0, 0.6, 0.8, 1).$$

Since $t = 2 \in T^{\square}$ as well, we repeat this computation:

$$P(t = 1) = P(t = 2) \cdot (M^+)^T = (0.8, 0.92, 0.96, 1).$$

Finally, since $t = 1 \notin T^{\square}$, we get

$$P(t = 0) = P(t = 1) \cdot (M^-)^T = (0.96, 0.864, 0.928, 1)$$

This vector contains, for each state, the probability that an object started at this position will intersect the query. Let us again, assume that initially, the object is located at s_2, i.e. $P(o, 0) = (0, 1, 0, 0)$ we finally obtain:

$$P^{\exists}(o, S^{\square}, T^{\square}) = P(o, 0) \cdot P(t = 0)^T = 0.864.$$

This result equals the result that we derived using the object-based approach.

12.3.3 Discussion

The advantage of the query-based approach is that we only have to compute $P(t = 0)$ once, and then compute, for each object o the $P^{\exists}(o, S^{\square}, T^{\square})$ by one single vector multiplication, which can be performed in $O(|P(o, 0)|)$, where $|P(o, 0)|$ is the number of non-zero elements in $P(o, 0)$, i.e. the number of possible positions of o at $t = 0$. In particular, if we assume that the number of possible states observed at $t = 0$ is small (which is realistic even for inaccurate observation types),we approach a total CPU cost of $O(1)$ per object. The initial computation of $P(t = 0)$ has to perform time-transitions using the transposed Markov-Chain. Thus, a vector-matrix multiplication is required for each transition from $t^{\square}_{end}$ to $t = 0$. Thus, the total runtime of the query-based approach is $O(|\mathcal{D}| + |S_{reach}|^2 \cdot \delta t)$, where $|\mathcal{D}|$ is the number of database objects, $|S_{reach}|$ is the number of states that have to be explored, and δt is the number of transitions between $t = 0$ and $t^{\square}_{end}$.

In contrast, the object-based approach has to perform time-transitions using the Markov-Chain *for each object.* Although it is possible in some cases to stop these transitions early using the inherent true-hit detection (computation can be stopped as soon as the probability of state ✓ becomes sufficiently large), in the worst case, all transitions from $t = 0$ to $t^{\square}_{end}$ have to be performed, and for each such transition, a vector-matrix multiplication has to be performed in $O(|S_{reach}|^2)$ time, where $|S_{reach}|$ is the total number of states reachable by o in the time interval $[0, t^{\square}_{end}]$. The total runtime of the object-based approach is thus $O(|\mathcal{D}| \cdot |S|^2 \cdot \delta t)$.

Nevertheless, the query-based approach makes the assumption that all objects follow the same model, i.e. all have the same Markov-Chain. In many applications, this assumption is reasonable: The movements of all icebergs are subject to the same currents - the form and shape of an iceberg can be assumed to have negligible impact on the icebergs movement. In road networks, objects may indeed follow different models. In the worst case, we may have to perform the query-based approach once for each object, to compute $P(t = 0)$ for it. If the objects can be partitioned to classes (e.g. buses, trucks and cars), the query-based approach can naturally be applied once for each class. In general, if objects follow different Markov-Chains, a technique to speed up the query-based approach is to cluster objects with similar Markov-Chains, and represent each cluster by one approximated Markov-Chain, where each entry is a probability interval instead of a singular probability. This approximated Markov-Chain can be used to perform pruning by detecting clusters of objects which must have (or cannot possibly have) a sufficiently high probability to satisfy the query predicate. Only clusters which cannot be decided as a whole need their objects to be considered individually.

12.4 Multiple Observations

So far, we have assumed that there is only one observation per object and that this observation happened at a time that (temporally) preceded the query time. In this section, we first show how to compute $P^{\exists}(o, S^{\square}, T^{\square})$ given two observations, one before query time and one after query time. Abstractly, this approach can be seen as a time-interpolation, whereas so far we have only considered time-extrapolation. The aim is to incorporate knowledge of both observations, in order to exclude all worlds which are not possible, given both observations, and properly re-weight the probabilities of the remaining worlds. Our proposed technique is applicable for both the object-based as well as the query-based approach. Later, we will give an intuition why considering additional observations, which are further apart from the query window than a considered observation, may still provide

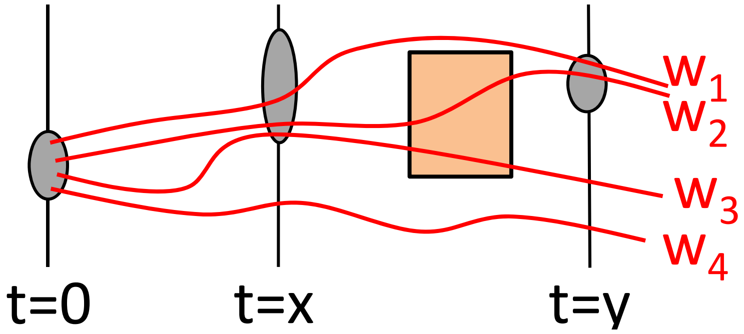


Figure 12.5: Multiple observations of an object

additional information. Finally, we outline how these techniques can be easily adapted to incorporate information from an arbitrary number of observations.

Given multiple observations, we have to distinguish between three classes of worlds:

A Worlds that become impossible due to the given observations,

B possible worlds that satisfy the query predicate and

C possible worlds that do not satisfy the query predicate.

The example in Fig. 12.5 shows 4 possible worlds (trajectories) of an object o observed at a point of time $t = 0$. In addition, two further observations of o exists at time $t = x$ and $t = y$. The possible locations of o at each time of the observations are illustrated by the ellipses. Due to the observation made at $t = y$, worlds w_3 and w_4 become impossible because they include impossible states at time $t = y$, i.e. both worlds belong to class A. In contrast, w_2 belongs to class B, since this world includes only possible states at all three observations and satisfies the query predicate of intersecting the query window. Finally, w_1 belong to class C, since, albeit possible, it does not intersect the window.

Intuitively and according to the possible worlds semantics, the probability P_{total} that an object satisfies the query predicate, given some observations, is the fraction of possible worlds that satisfy the query predicate, i.e. the fraction

$$P_{total} = \frac{P(B)}{P(B) + P(C)}. \tag{12.1}$$

For each object $o \in \mathcal{D}$ we assume a set of observations $\Theta^o = \{< t_1^o, \theta_1^o >, < t_2^o, \theta_2^o >, \ldots, < t_{|\Theta^o|}^o, \theta_{|\Theta^o|}^o >\}$ where $t_i^o \in \mathcal{T}$ denotes the time and $\theta_i^o \in \mathcal{S}$ the location of an observation. W.l.o.g. let $t_1^o < t_2^o < \ldots < t_{|\Theta^o|}^o$. The location of an observation is given by a probability density function over the state space. In our case of discrete space, this PDF can be seen as a $|\mathcal{S}|$-dimensional probability vector, i.e. $\theta_i^o \in \mathbb{P}^{|\mathcal{S}|}$, with $\mathbb{P} = [0, 1]$.

Then, the derived matrices M^- and M^+ can be used for the query-based approach. The approach of Section 12.3.1 cannot be applied directly, because now, worlds which have reached the query window are no longer equivalent, and can no longer be unified in a single state ✓. The reason is that the current state of such a world now effects the probability of reaching the state observed at time $t = 3$. Therefore, for each world that has intersected the query, we need to maintain information about its current state at each time. Therefore, we replace the state ✓ by a set of states s_1✓,...,s_3✓. Each state s_i✓ corresponds to the probability, that o has intersected the query window *and* is currently located in state s_i. The transition matrices M^- and M^+ need to be adjusted accordingly. The shape of M^- is clear: For a transition from t to $t+1$, where $t+1 \notin T^\square$ (the case where M^- is used), states are simply transitioned, and worlds which have (not) intersected the query at t must (not) have done so at $t+1$. We obtain:

$$M^- = \begin{pmatrix} M & 0 \\ 0 & M \end{pmatrix}$$

In the case where M^+ is used, that is the case where $t+1 \in T^\square$, we have to ensure that any transition to a state $s \in S^\square$ leads to the corresponding state s✓. This yields the matrix

$$M^+ = \begin{pmatrix} M - M' & M' \\ 0 & M \end{pmatrix},$$

where M' is derived from M by setting all columns to zero, where the corresponding state $s \notin S^\square$, and $M - M'$ is derived by setting all columns to zero for which $s \in S^\square$.

We start by using the probability distribution θ_1^o of an object o observed at t_1^o. As in the previous sections, we iteratively apply the modified matrices M^- and M^+ until we reach t_2^o. At this point, we have two probability distributions: One distribution derived using the observation at t_1^o, which has been transitioned to t_2^o, as well as the probability distribution θ_2^o. These observations can be unified by exploiting independence between observations.

Lemma 12.1. Let $X(i) := \{\theta_{i,1}^o, \ldots, \theta_{i,n}^o\}$ be a set of PDFs of an object o at time t_i, derived from independent observations. The joint probability distribution $P(o,t)$ of o at time t is given by:

$$P(o, t_i^o) = N(\prod_{x \in X} x_1, ..., \prod_{x \in X} x_{|\mathcal{S}|}),$$

where $N(\cdot)$ is the vector normalization function, i.e. $N(x) = (\frac{x_1}{\sum x}, ..., \frac{x_{|\mathcal{S}|}}{\sum x})$

Proof. For each $j \in 1, ..., |\mathcal{S}|$, $P(o, t_i)_j$ is, by definition of a probability density function, the probability of the random event that object o is located in state s_j. Without loss of generality, let $a = (a_1, ..., a_{|\mathcal{S}|}) \in X$ be the first observation. Given this observation only, then clearly $P(o, t_i) = a$ holds. Given further observations, the probability of this event becomes conditioned to

$$P(o,t)_j = P(a_j | X \setminus \{a\}).$$

Since the errors of all observations are mutually independent, we get

$$P(o, t_i)_j = a_j \cdot \prod_{x \in X} x_j,$$

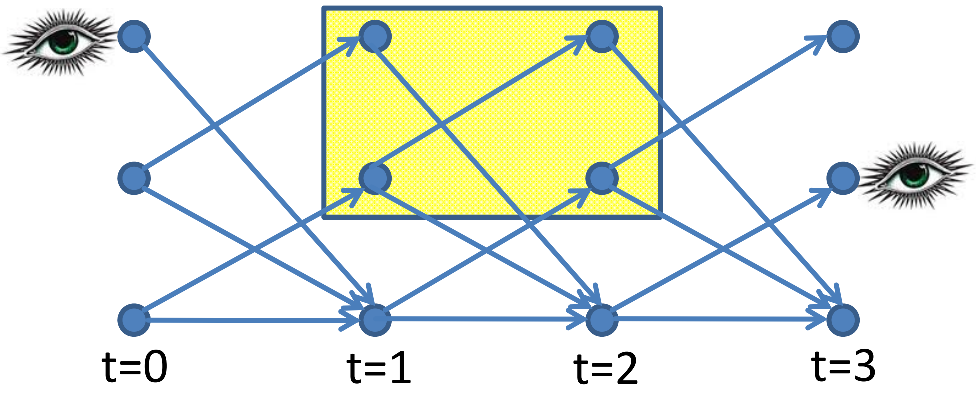


Figure 12.6: Two observations of an object

which is the fraction of all worlds, including worlds which are no longer possible given the observations X, in which o is located in state s_j at time t_i. Due to possible worlds semantics (c.f. Equation 12.1), we are only interested in the fraction of possible worlds in which o is located in state s_j at time t. Since v contains all possible worlds, the normalization $N(v)$ yields the correct result. □

As an example, consider Figure 12.6, where we again use our running example Markov-Chain

$$M = \begin{pmatrix} 0 & 0 & 1 \\ 0.5 & 0 & 0.5 \\ 0 & 0.8 & 0.2 \end{pmatrix}$$

and assume that an object O has been observed at state s_1 at time t_1 and at state s_3 at time t_4. We obtain the transition matrices

$$M^- = \begin{pmatrix} M & 0 \\ 0 & M \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0.5 & 0 & 0.5 & 0 & 0 & 0 \\ 0 & 0.8 & 0.2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0 & 0 & 0.8 & 0.2 \end{pmatrix}$$

and

$$M^+ = \begin{pmatrix} M - M' & M' \\ 0 & M \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0.5 & 0 & 0 \\ 0 & 0 & 0.2 & 0 & 0.8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0 & 0 & 0.8 & 0.2 \end{pmatrix}$$

Since at time $t = 0$, o has been observed in state s_1, and since o cannot have reached the window yet, the initial distribution of o is $\theta_1^o = P(o, 0) = (1, 0, 0, 0, 0, 0)$. Transition to $t = 1$

using M^+ (since $t = 1 \in T^\square$) yields $P(o, 1) = (0, 0, 1, 0, 0, 0)$. The next transition using M^+ yields $P(o, 2) = (0, 0, 0.2, 0, 0.8, 0)$. The intuition of this vector is that, at time $t = 2$, object o is located in state s_3 while having reached the query window with a probability of 20%, and otherwise is located in state s_2 having reached the query window. The next transition uses M^- (since $t = 3 \notin T^\square$) and yields $P(o, 3) = (0, 0.16, 0.04, 0.4, 0, 0.4)$. Now, at time $t = 3$, the second observation was made. We assume that this observation only has information about the state at $t = 3$, but no information whether o has intersected the query window. Thus, the observation vector has the form $\theta_2^o = (0, 0.5, 0, 0, 0.5, 0)$. Due to Lemma 12.1, and since we assume that the observation at t_1 (from which $P(o, 3)$ was derived) and θ_2^o are independent observations, we can directly multiply the entries of $P(o, 3)$ and θ_2^o, yielding $P(o, t)\prime = (0, 0.08, 0, 0, 0, 0)$. Normalization yields $P(o, t) = N(P(o, t)\prime) = (0, 1, 0, 0, 0, 0)$, which means that at $t = 3$, given both observations, o must be in state s_2 and must not have intersected the query window. This is intuitive, since the only path between s_1 at $t = 0$ and s_2 at $t = 3$ does not intersect the query window.

12.5 Additional Spatio-Temporal Queries

Based on the proposed concepts for answering spatio-temporal ∃-queries, we will now show how different query predicates can be answered efficiently.

PST∀Q:

In some applications, it may be interesting to compute the probability that o is in the query window at *all* times $t \in T^\square$. Clearly, the probability that o is located in $S^\square$ at all times in $T^\square$ complements the probability that o is outside $S^\square$ at any time in $T^\square$, i.e.

$$P^\forall(o, S^\square, T^\square) = 1 - P^\exists(o, \mathcal{S} \setminus S^\square, T^\square).$$

Although, in general, $|\mathcal{S}| >> |S^\square|$, the time required to compute $P^\exists(o, \mathcal{S} \setminus S^\square, T^\square)$ is generally not larger than the time required to compute $P^\forall(o, S^\square, T^\square)$. The only difference between these two computations is the content of the matrix M^+, since different sets of columns of matrix M are merged. In most cases, the computation of $P^\exists(o, \mathcal{S} \setminus S^\square, T^\square)$ is actually faster, since more columns of M^+ are zero.

PSTkQ:

So far, an object o satisfies the query predicate in any world where it intersects the query window, regardless for how long o remains in the query window. In the following, we will show how the probability distribution of the number of times that o is located in the query window (c.f. Definition 12.4) is computed.

To answer this query, we can extend the idea of adding new virtual states, to capture the number of times an object has visited the query window; we use the set of states $\mathcal{S}' = \mathcal{S} \times \{0, ..., |T^\square|\}$. Intuitively, an object in state $s' = (s \in \mathcal{S}, k \in \{0, ..., |T^\square|\}) \in \mathcal{S}'$

is currently located in state s and has been in the query window at k points of time. At any point of time $t \in T^\square$, all possible worlds located at a state $s \in S^\square$ are transitioned in order to increase their k value by one. The resulting matrices are

$$M^- = \begin{pmatrix} M & 0 & ... & 0 \\ 0 & M & ... & 0 \\ ... & ... & ... & ... \\ 0 & 0 & ... & M \end{pmatrix}$$

$$M^+ = \begin{pmatrix} M - M' & M' & 0 & ... & 0 \\ 0 & M - M' & M' & ... & 0 \\ ... & ... & ... & ... & ... \\ 0 & 0 & ... & M - M' & M' \end{pmatrix}$$

Since initially, an object o visited the query window zero times[3], the initial distribution of an object is concatenated with $k \cdot |\mathcal{S}|$ zeroes. If we perform time transitions until we reach $t^\square_{end}$, we obtain for each $s' = (s \in \mathcal{S}, k \in \{0, ..., |T^\square|\}) \in \mathcal{S}'$ the probability that o will visit the query window exactly k times and will finally be in state s. Grouping this result by k, we obtain for each $k \in \{0, ..., |T^\square|\}$, the probability that o visits the query window exactly k times.

Obviously the above approach is not very memory efficient, since M^- and M^+ each blow up the memory requirement of the original transition matrix M by a factor of $|T^\square|$. Thus, we suggest a more memory efficient algorithm for this problem. For processing an object $o \in \mathcal{D}$, an additional $(|T^\square| + 1) \times |\mathcal{S}|$-Matrix $C(t)$ is necessary. Each entry $c_{i,j}(t) \in C(t)$ corresponds to the probability that o is currently located in state s_j and has been located in $S^\square$ at exactly i points of time $t' \leq t \in T^\square$. We begin by setting the first row of $C(0)$ to $P(o, 0)$ and all other entries to zero, since at $t = 0$, o cannot possibly have entered the query region. At each state transition from t to $t + 1$, a transition using M is performed for each row. This is simply achieved by computing $C'(t + 1) = C(t) \cdot M$. If t is not in $T^\square$, then we set $C(t+1) = C'(t+1)$. Otherwise, if $t \in T^\square$, then we additionally shift each column corresponding to a state $s_i \in \mathcal{S}$ down by one, and fill the top entry of this line with zero. That is

$$c_{i,j} = \begin{cases} c'_{i,j}, & \text{if } j \notin S^\square \\ 0, & \text{if } j \in S^\square \wedge i = 0 \\ c'_{i-1,j} & \text{otherwise.} \end{cases}$$

When $t^\square_{end}$ is reached, the probability for o to be in the query exactly k-times can be obtained by summing up all probabilities in row k of $C(t^\square_{end})$. Thus the probability $P^{k-times}(o, S^\square, T^\square)$ that o has visited the query window exactly k times is given by

$$P^{k-times}(o, S^\square, T^\square) = \sum_j c_{k,j}(t^\square_{end})$$

---

[3] The special case where $t = 0$ belongs to $\mathcal{T}^\square$ is omitted. In that case, the initial distribution of an object simply starts at $k = 1$, i.e. is shifted by $|\mathcal{S}|$.

Considering our running example we start with the matrix $C(0)$ and transition as along as $t \notin T^{\Box}$:

$$C(0) = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \xrightarrow{\cdot M^2} C(2) = \begin{pmatrix} 0 & 0.32 & 0.68 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Now we shift down the probabilities of the states in $S^{\Box}$ by one row:

$$C(2) = \begin{pmatrix} 0 & 0 & 0.68 \\ 0 & 0.32 & 0 \\ 0 & 0 & 0 \end{pmatrix} \xrightarrow{\cdot M} C(3) = \begin{pmatrix} 0 & 0.544 & 0.136 \\ 0.192 & 0 & 0.128 \\ 0 & 0 & 0 \end{pmatrix}$$

After performing the last shift we obtain

$$C(3) = \begin{pmatrix} 0 & 0 & 0.136 \\ 0 & 0.544 & 0.128 \\ 0.192 & 0 & 0 \end{pmatrix} \xrightarrow{rowsum} \begin{pmatrix} 0.136 \\ 0.672 \\ 0.192 \end{pmatrix}.$$

The resulting vector reflects the probability of o to be in the query exactly 0 (0.136), 1 (0.672) and 2 (0.192) times.

12.6 Experimental Evaluation

12.6.1 Datasets and Experimental Setting

For our experimental evaluation we used synthetic as well as real datasets. In order to observe the influence of data characteristics we used several parameters for the construction of the synthetic datasets. Each experiment is performed using a database of $|\mathcal{D}|$ objects. The location of each object at time t_0 is given by a PDF over a certain number of states. This value is controlled by the parameter *object_spread* and characterizes the amount of uncertainty in the database. The total number of states of the system is characterized by $|\mathcal{S}|$. To control the density of the transition matrix (which corresponds to the number of possible transitions in the modeled system) we used the parameter *state_spread*. From each state it is possible to transition into *state_spread* states. To model locality of the transitions within the system we also introduced a parameter which bounds the possible states which can be reached by one transition. An object in state s_i can only transition into states $s_j \in [s_{i-max\_step/2}; s_{i+max\_step/2}]$. All parameters for the synthetic datasets are summarized in Table 12.2.

As real datasets we used two road network datasets. The first is the road network of North America which consists of 175,813 nodes and 179,102 edges. As this is a rather sparse graph we also extracted the road network from Munich which has 73,120 nodes and 93,925 edges. The transition matrix is equivalent to the adjacency matrix of the corresponding graph. This means each node is treated as a state and each edge corresponds to two non-zero entries in the transition matrix. The value of the non-zero entries of one line in

parameter	**value range**	**default**
$\|\mathcal{D}\|$	1,000 - 100,000	10,000
$\|\mathcal{S}\|$	2,000 - 100,000	100,000
object_spread	5	5
state_spread	1 - 20	5
max_step	10 - 100	40

Table 12.2: Parameters for the synthetic datasets

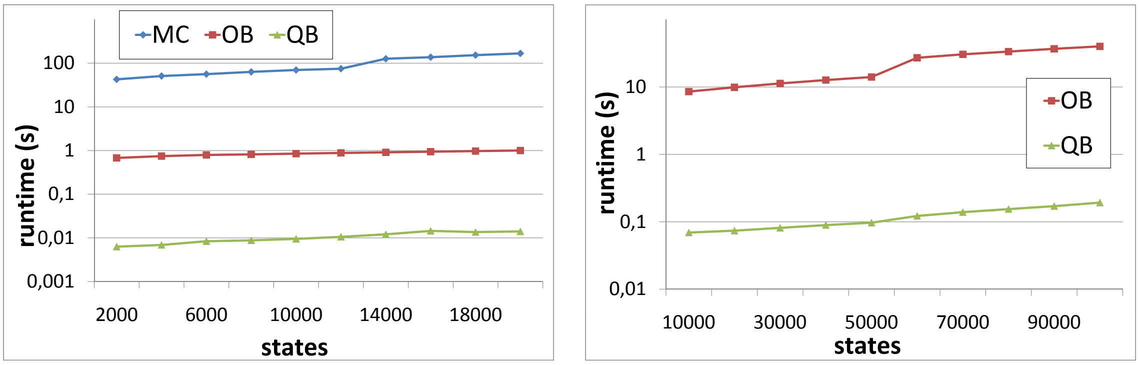
the matrix are set randomly and sum up to one. In this way they reflect the transition probabilites from one node in the road network to its directly connected neighbors.

In our experiments we evaluate object-based (**OB**) and query-based (**QB**) query processing using several query predicates ($\exists, \forall$ and k-count). Additionally we compare these approaches to a Monte-Carlo based method (**MC**). The **MC** approach samples paths of each object and outputs the fraction of the sampled paths which fulfill the query predicate. Sampling the path of an object requires first drawing a start state from the objects distribution. Afterwards for each timestep a state from the successor states of the current state is chosen according to the probability distribution given by the transition matrix. Note that **MC** only returns approximate results, where the accuracy can be improved if more paths are sampled. Since the sampling of paths is equivalent to a Bernoulli sequence, the standard deviation between the sampled probability ($\hat{p}$) and the true probability (p) is given by $\sigma_{\hat{p}} = \sqrt{\frac{p\cdot(1-p)}{n}}$. For 100 samples, the standard deviation between p and $\hat{p}$ is thus at least 5% and gets worse for small and large values of p.

All experiments were run on a single 64-Bit machine with an Intel Xeon 5160 processor with 3.0 GHz and 32GB of RAM. The computations where performed using MATLAB R2011a. Unless mentioned otherwise, we generated 10,000 objects randomly distributed across the state space and the query window is defined by the states [100, 120] and time interval [20, 25]. For the Monte-Carlo based approach the number of drawn samples was set to 100.

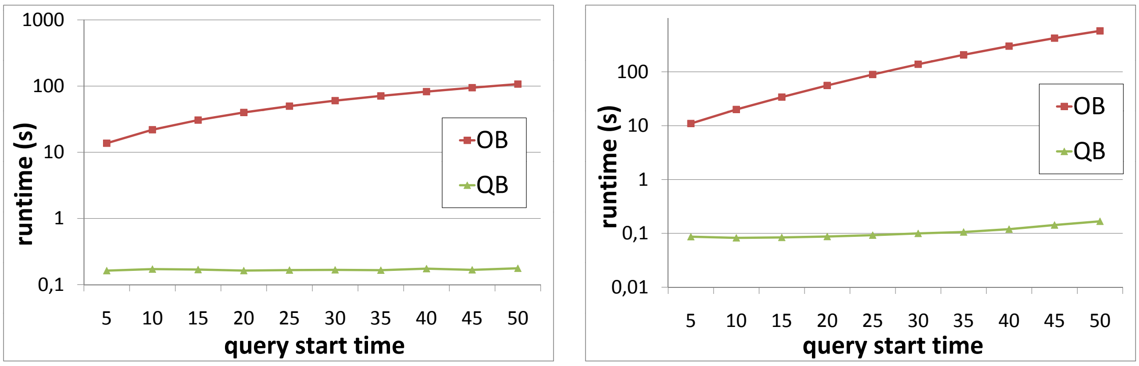
12.6.2 Experiments

In the first experiment, we vary the size of the state space $|\mathcal{S}|$ and measure the cost of query evaluation for a PST$\exists$Q. In Figure 12.7(a), we used a relatively small synthetic dataset ($|\mathcal{D}| = 1,000$, $|\mathcal{S}| = [1,000; 10,000]$). The **MC** approach is computationally very demanding in comparison to the other two algorithms. The reason is, that even for such a small setting the Monte-Carlo based approach has to draw a very high amount of samples. Note that already for a small number of sampled paths (we used 100 in this setting) this approach becomes expensive, because the sampling of one path already requires to draw as many samples as the considered stamps in the query. This corresponds to 2,500 samples for one object in the database. Due to these high costs we excluded the **MC** algorithm from the remaining experiments. As expected the **QB** approach is much faster

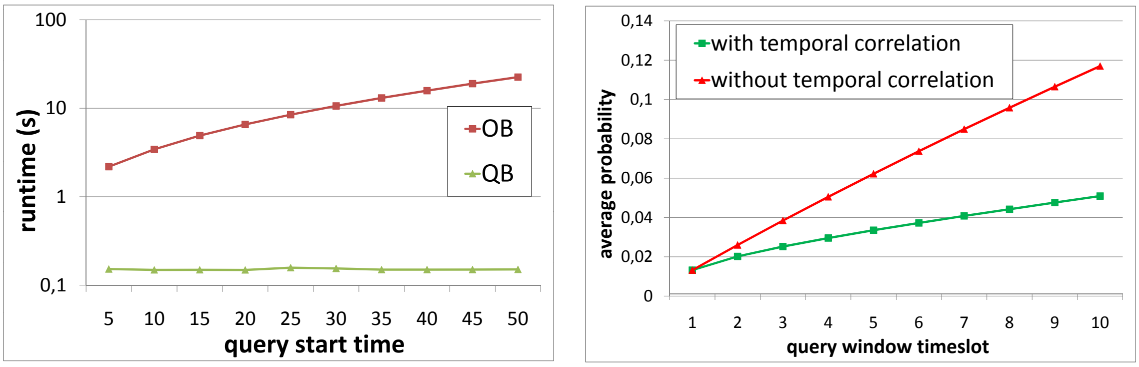


(a) Small state space (b) Large state space

Figure 12.7: Query processing runtime w.r.t. the number of states.

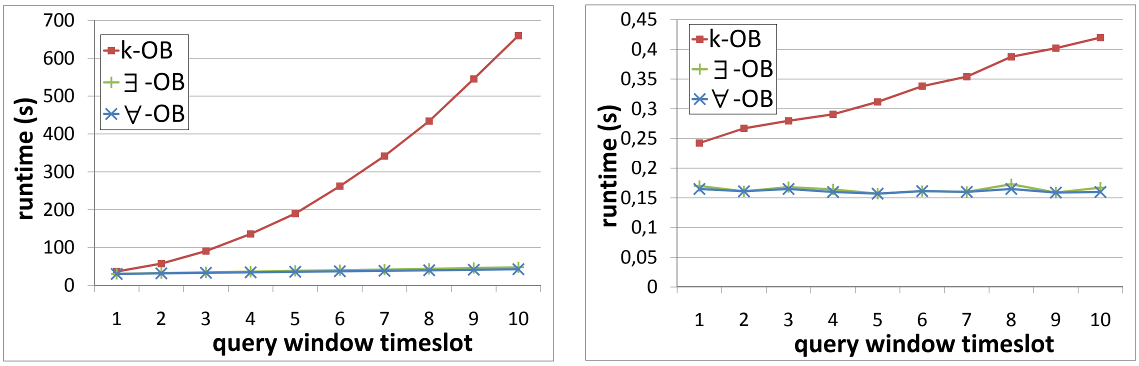


(a) Synthetic data (b) Munich dataset



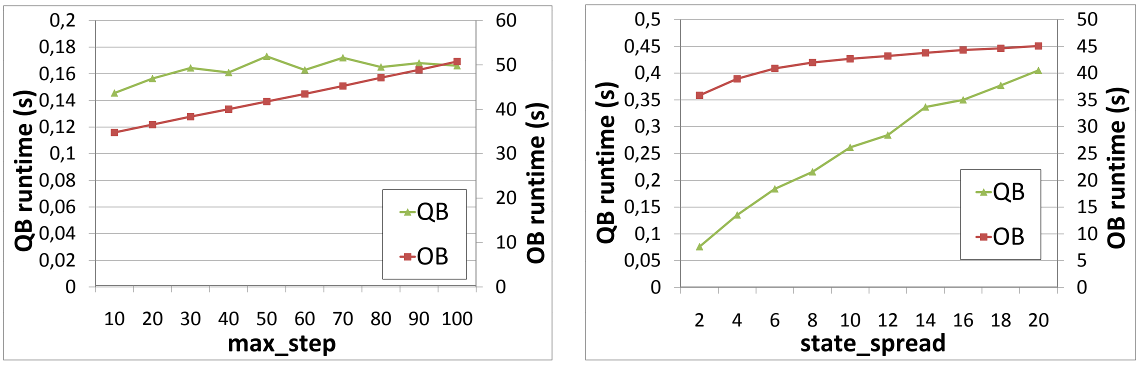
(c) NA dataset (d) accuracy experiment

Figure 12.8: Query processing runtime w.r.t. the size of the query time frame



(a) OB approach (b) QB approach

Figure 12.9: Performance evaluation of query predicates.



(a) Impact of max_step (b) Impact of state_spread

Figure 12.10: Comparison of QB and OB behavior with scaling parameters

than the **OB** approach. Figure 12.7(b) shows how these two methods scale at larger datasets ($|\mathcal{D}| = 100,000$, $|\mathcal{S}| = 10,000$).

The experiments shown in Figures 12.8(a) and 12.8(b) show the dependency of the PST∃Q algorithms on the timeslot we want to query on synthetic and real data sets. The runtime of **OB** is increasing much faster than the runtime of **QB**. As expected the runtimes of both algorithms suffer from a longer glance in the future as the vectors to be multiplied become less sparse with each time stamp. Besides this, **QB** should not be influenced by the number of timestamps whereas the runtime of the **OB** approach scales linearly to that parameter. To justify the used model, we also performed an experiment which compared our model to a model where temporal correlations are ignored (cf. Figure 12.8(d)) w.r.t. accuracy. In this experiment we posed PST∃ queries with an increasing query time window and measured the average probability of objects having a non-zero probability to fulfill the query predicate. It is clearly visible that ignoring the temporal dependency returns a biased result and the error compared to the correct result even increases for larger query windows.

In Figure 12.9(a) we compare the three proposed query types PST∃Q, PST∀Q and PSTkQ query using the object-based approach. Obviously for the PSTkQ we have to maintain not only one but multiple vectors (as many as the number of times in $T^{\square}$) per object, which leads to an increased runtime. The PST∃Q and PST∀Q had equal runtime in all experimental settings. Using the **QB** approach all queries run in a fraction of a second and the runtime of PSTkQ seems to scale rather linearly with k (cf. Figure 12.9(b)) .

In the next set of experiments, the runtime behavior of the two approaches w.r.t. different locality parameters is tested. Figure 12.10(a) shows the runtime for increasing the *max_step* parameter whereas Figure 12.10(b) shows results for increasing *state_spread* parameter (Note that the algorithm runtimes are marked at different axes). Both algorithms scale at most linearly with those parameters.

12.7 Conclusion

In this chapter, we studied the problem of probabilistic query evaluation over uncertain spatio-temporal data. We consider uncertain trajectories, for which some points are sampled via observations, while the remaining points are instantiated by a stochastic process. We investigated query processing over uncertain moving object data, which are modeled by stochastic processes, specifically Markov-Chains. This approach has three major advantages over previous work. First it allows answering queries in accordance with the possible worlds model. Second, dependencies between object locations at consecutive points in time are taken into account. And third, it allows us to infer the probability an object reaches a certain location (state) by matrix multiplications that can be processed extremely efficient. Based on this method we propose a framework for processing queries over such data, which injects pruning techniques into the Markov Chain matrices. An object-based and a query-based approach are proposed; the latter is always more efficient, typically by orders of magnitude. In our experiments we show that we are able to answer queries on settings with 100,000 location states and 10,000 objects in a fraction of a second on a single machine in contrast to state-of-the-art solutions that are multiple orders of magnitude slower, e.g. the Monte-Carlo approach requires several hours for the same query. We show how the framework can be applied for the cases where there exist one or multiple observations per object and for various probabilistic spatio-temporal query variants.

Chapter 13

Indexing Uncertain Spatio-Temporal Data

In the last chapter we showed how to model uncertain spatio-temporal data and perform probabilistic spatio-temporal window queries on this model. Though this proceeding showed to be very efficient in terms of CPU time, it is still necessary to evaluate each object in the database separately resulting in a sequential scan through the whole database. For large amounts of uncertain spatio-temporal data this may not be feasable especially in time critical applications. In this chapter, we propose an index structure that facilitates efficient processing of spatio-temporal window queries based on an appropriate model for uncertain trajectories. Our hierarchical index uses novel approximation techniques in order to probabilistically bound the uncertain movement of objects; these techniques allow for efficient and effective filtering during query evaluation. As underlying model for the movement of each object we assume a Markov Chain. However the pruning techniques of the index structure are not restricted to this model. The objective of our index is to minimize the number of objects for which exact probabilistic evaluation has to be performed.

Therefore, we show how to bound the movement of each object o and derive a *spatio-temporal approximation* of o. For example, Figure 13.1 shows that the movement of the object between consecutive observations can be bounded by a diamond, assuming that the object's velocity is bounded. Furthermore, we propose techniques to identify *probabilistic spatio-temporal approximations*, i.e. smaller spatio-temporal approximations which guar-

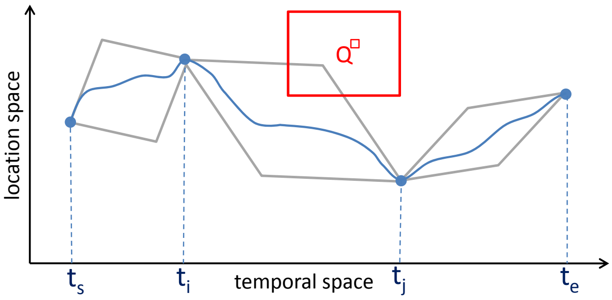


Figure 13.1: Spatio-temporal data.

antee that an object remains inside them with a certain probability. By indexing both types of approximations for all objects, we can prune objects that do not qualify a given probabilistic query. Parts from this chapter have been published in [65].

The rest of the chapter is organized as follows. Section 13.1 presents the model that we adopt for the uncertain movement of an object between consecutive observations and formally defines the spatio-temporal queries that we address in our study. Related work is discussed in Section 13.2. Section 13.3 presents our approximation techniques for object trajectories, based on which the proposed index is developed (cf Section 13.4). Section 13.5 presents our experimental evaluation. Finally, Section 13.6 concludes the chapter.

13.1 Problem Definition

This section formally defines the type of spatio-temporal data that we index, the stochastic model that we use for uncertain trajectories derived from the data, and the query types that we handle.

Data: We consider a discrete time ($\mathcal{T} = \mathbb{N}_0^+$) and space ($\mathcal{S} = \{s_1, ...s_{|\mathcal{S}|}\} \subseteq \mathbb{R}^D$) domain as assumed in the previous Section and many existing works (e.g. [135, 12, 79]). Given this spatio-temporal domain, the (certain) movement of an object o corresponds to a *trajectory* represented as function $o : \mathcal{T} \rightarrow \mathcal{S}$ of time defining the location $o(t) \in \mathcal{S}$ of o at a certain point of time $t \in \mathcal{T}$. We consider incomplete (and/or imprecise) spatio-temporal data, where the motion of an object is not recorded by a crisp trajectory. Instead, we are only given a set Θ^o of *(time, location)* observations for each object o. At any time $t \notin \Theta^o$, the position of o is uncertain, i.e., a random variable. Note that in this chapter we assume certain observations, i.e. the location distribution $\theta_i^o \in \Theta^o$ at time t_i^o concentrates in one state. In other words θ_i^o is a distribution vector with all-zero entries, except for one entry which is equal to 1 (the state corresponding to the certain location of the observation). In the scope of this chapter and since the context is clear we use θ_i^o also to refer to the state of the observation Θ_i^o. Thus for each observation Θ_i^o it holds that $P_{\theta_i^o}(o, t_i^o) = 1$.

Uncertain Data Model: We refer to the *spatio-temporal approximation* of a trajectory of an object o in a time interval spanned by two consecutive observations of o at t_i^o and t_j as a *bead* or *diamond* $\diamond(o, t_i^o, t_j)$. The diamond can be computed by considering the maximum and minimum (singed) velocities of the object in each dimension [161]. The whole approximation of the trajectory based on a set Θ^o of observations (i.e., a chain of beads) is referred to as a *necklace*. For example, the movement of the object in Figure 13.1 is bounded by a chain of diamonds.

Again we model the uncertain movement of an object within a diamond, using a first-order Markov-chain model (cf Definition 12.5) as described in Section 12.2. In this chapter, however we consider the case where each object may have its own time-dependent transition matrix $M^o(t)$.

Queries: Within the scope of this chapter, we focus on selection queries specified by the following parameters: (i) a spatial window $S^\square \subseteq \mathcal{S}$, (ii) a contiguous time window $T^\square \subseteq \mathcal{T}$, and (iii) a probability threshold τ. In the remainder, we use $Q^\square = S^\square \times T^\square$ to denote the search space of a query. Specifically we will consider probabilistic spatio-temporal τ exists (PST$\tau\exists$) queries and spatio-temporal τ forall (PST$\tau\forall$) queries. A PST$\tau\exists$ query returns all objects $o \in \mathcal{D}$ for which $P^\exists(o, S^\square, T^\square) \geq \tau$ (cf Definition 12.2). Analogously a PST$\tau\forall$ query returns all objects $o \in \mathcal{D}$ for which $P^\forall(o, S^\square, T^\square) \geq \tau$ (cf Definition 12.3). By modeling the movement within a diamond using a Markov-chain model, the true probability that an object o satisfies a PST$\tau\exists$ or PST$\tau\forall$ query, can be computed in PTIME as shown in Section 12.3.3, exploiting that the matrix M is generally sparse (only a few states are directly connected to a single state). Still, query evaluation remains too expensive over a large spatio-temporal database, where we have to compute the qualification probabilities of all objects. In view of this, we define a set of approximations of uncertain object trajectories enabling spatio-temporal and probabilistic filtering in Section 13.3. We then show in Secion 13.4 how we can organize these approximations in an index in order to perform efficient query evaluation.

13.2 Related Work

The problem of managing, mining and querying spatio-temporal data has received continuous attention over the past decades (for a comprehensive coverage, see [76]). Specifically for efficient query processing a vast amount of indexing structures for different purposes and data characteristics has been developed (an overview and a classification can be found in [120] and [123]). From this body of work, our approach is mostly related to spatio-temporal data indexing for predictive querying, for example indexes like [137, 89] and approaches like [149]. Still, these papers neither consider probabilistic query evaluation nor model the data with stochastic processes.

Our work is also inspired by methods for indexing uncertain spatial data. The U-Tree [149, 155] and its extension [179] bound each spatial uncertain object with an MBR and additionally associate it with a set of "probabilistically constrained regions" (PCR). These PCRs can be used for probabilistic pruning during query processing. For efficiency reasons the set of PCRs are conservatively approximated by a linear function over the parameters of the PCRs.

13.3 Approximating UST-Objects

In this section, we introduce (conservative) spatio-temporal as well as (conservative) probabilistic spatio-temporal (UST-) object approximations, which serve as building blocks for our proposed index, to be presented in Section 13.4.

13.3.1 Spatio-Temporal Approximation

To bound the possible locations of an object o between two subsequent observations $(\Theta_i^o, \Theta_{i+1}^o)$, we need to determine all state-time pairs $(s,t) \in S \times T, t_i^o \leq t \leq t_{i+1}^o$ such that o has a non-zero probability of being at state s at time t. This is done by considering all possible paths between state θ_i^o at time t_i^o and state θ_{i+1}^o at time t_{i+1}^o. An example of a small set of such paths is depicted in Figure 13.2(a). Here, we can see a set of five possible trajectories of an object o, i.e., all possible $(state, time)$ pairs of o in the time interval $[t_i^o, t_{i+1}^o]$. In practice, the number of possible paths becomes very large. Nonetheless, we can efficiently compute the set of possible $(state, time)$ pairs using the Markov-chain model: The set of state-time pairs S_i reachable from Θ_i^o can be computed by performing $t_{i+1}^o - t_i^o$ transitions using the Markov chain $M^o(t)$ of o, starting from state θ_i^o and memorizing all reachable $(state, time)$ pairs. Similarly, we can compute S_{i+1} as all state-time pairs $(s, t) \in \mathcal{S} \times \mathcal{T}, t_i^o \leq t \leq t_{i+1}^o$ such that o can reach state θ_{i+1}^o at time t_{i+1}^o by starting from state s at time t. S_{i+1} can be computed in a similar fashion, starting from state θ_{i+1}^o and using the transposed Markov chain $M^o(t)^T$. The intersection $S_{i,i+1} = S_i \cap S_{i+1}$ yields all state-time pairs which are consistent with both observations. Let us note that in practice, it is more efficient to compute S_i and S_{i+1} in a parallel fashion, to reduce the explored space. When the computation of S_i and S_{i+1} meet at some time $t_i^o \leq t \leq t_{i+1}^o$, we can prune any states which are not reachable by both θ_i^o at time t_i^o and θ_{i+1}^o at time t_{i+1}^o. However, the number $|S_{i,i+1}|$ of possible state-time pairs in $S_{i,i+1}$ can grow very large, so it is impractical for our index structure (proposed in Section 13.4) to store all $S_{i,i+1}$ for each $o \in DB$ in our index structure. Thus, we propose to conservatively approximate $S_{i,i+1}$. The issue is to determine an appropriate approximation of $S_{i,i+1}$ which tightly covers $S_{i,i+1}$, while keeping the representation as simple as possible. The basic idea is to build the approximation by means of both object observations Θ_i^o and Θ_{i+1}^o with the corresponding velocity of propagation in each dimension. To do so, we first compute for the set of state-time pairs S_i to derive the maximum and minimum possible velocity in the time interval $[t_i^o, t_{i+1}^o]$:

$$v_d^{\leqslant} := max_{(s,t) \in S_i}(\frac{s[d] - \theta_i^o[d])}{t - t_i^o})$$

$$v_d^{\leqslant} := min_{(s,t) \in S_i}(\frac{s[d] - \theta_i^o[d])}{t - t_i^o})$$

where $s[d]$ ($\theta_i^o[d]$) denotes the projection of state s (θ_i^o) to the d-th dimension. By definition, we can guarantee that for any $t_i^o \leq t \leq t_{i+1}^o$ it holds that

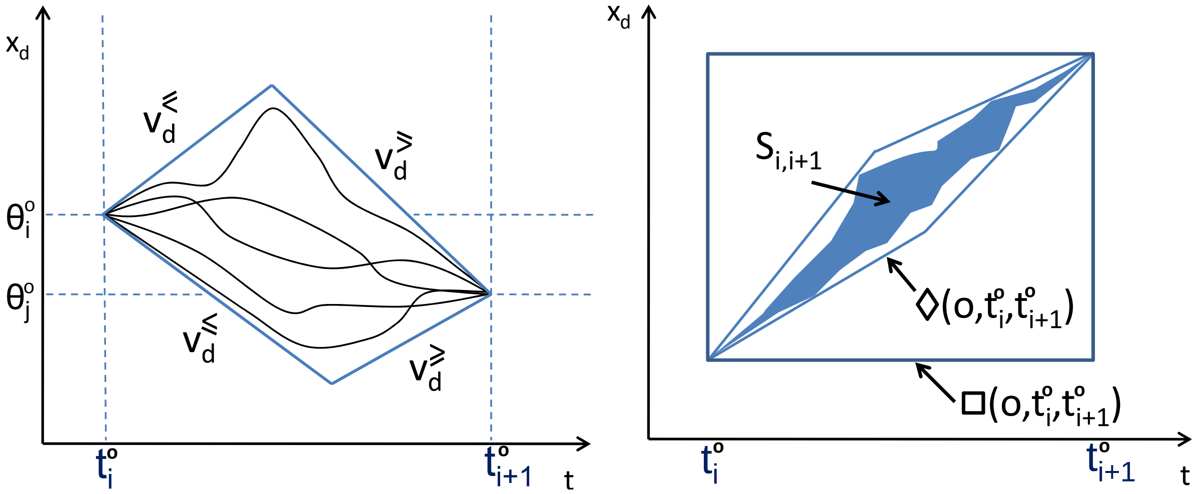
$$o(t)[d] \leq \theta_i^o[d] + (t - t_i^o) \cdot v_d^{\leqslant} \text{ and}$$

$$o(t)[d] \geq \theta_i^o[d] + (t - t_i^o) \cdot v_d^{\leqslant}$$

where $o(t)[d]$ denotes the projection of all states s with a non-zero probability $P(o(t) = s)$ on the d-th dimension. Furthermore, we bound the velocity of propagation at which o can have reached state θ_{i+1}^o at time t_{i+1}^o from each location in the state-space S_{i+1}:

$$v_d^{\geqslant} := max_{(s,t) \in S_{i+1}}(\frac{\theta_i^o[d] - s[d]}{t_{i+1}^o - t})$$

$$v_d^{\geqslant} := min_{(s,t) \in S_{i+1}}(\frac{\theta_i^o[d] - s[d]}{t_{i+1}^o - t})$$



(a) Approximating trajectories (b) Diamond vs MBR

Figure 13.2: Spatio-temporal approximation.

Again, we can bound the position of o in dimension $1 \leq d \leq D$ at time $t_i^o \leq t \leq t_{i+1}^o$ as follows:

$$o(t)[d] \leq \theta_{i+1}^o[d] - (t_{i+1}^o - t) \cdot v_d^{\geqslant}, \text{ and}$$
$$o(t)[d] \geq \theta_{i+1}^o[d] - (t_{i+1}^o - t) \cdot v_d^{\geqslant}$$

In summary, using the positions of state θ_i^o at time t_i^o and θ_{i+1}^o at time t_{i+1}^o, and using velocities $v_d^{\leqslant}, v_d^{\leqslant}, v_d^{\geqslant}, v_d^{\geqslant}$, we can bound the random variable of the position $o(t)$ of o at time $t_i^o \leq t \leq t_{i+1}^o$ by the interval

$$\begin{aligned} o(t)[d] \in I_d(t) := [&max(\theta_i^o[d] + (t - t_i^o) \cdot v_d^{\leqslant}, \theta_{i+1}^o[d] - (t_{i+1}^o - t) \cdot v_d^{\geqslant}), \\ &min(\theta_i^o[d] + (t - t_i^o) \cdot v_d^{\leqslant}, \theta_{i+1}^o[d] - (t_{i+1}^o - t) \cdot v_d^{\geqslant})] \qquad (13.1) \end{aligned}$$

Deriving these intervals for each dimension yields an axis-parallel rectangle, approximating all possible positions of o at time t. In the following, we will call this time dependent spatial approximation of $o(t)$ in the time interval $[t_i^o, t_{i+1}^o]$ between two observations θ_i^o and θ_{i+1}^o a spatio-temporal *diamond*, denoted as $\diamond(o, t_i^o, t_{i+1}^o)$ [1]. A nice geometric property of this approximation is that computing the intersection with the query window at each time t is very fast. Another advantage is that existing spatial access methods (e.g., R-trees) can be easily used to efficiently organize these approximations. To store the approximation, we only need to store the $4 \cdot D$ real values $v_d^{\leqslant}, v_d^{\leqslant}, v_d^{\geqslant}, v_d^{\geqslant}, 1 \leq d \leq D$. Note that the observations Θ_i^o and Θ_{i+1}^o which are furthermore required to span a diamond, are already stored in $\mathcal{D}$. A diamond is reminiscent to a time-parameterized rectangle, used to model the worst-case MBR for a set of moving objects in [137]; however, the way of deriving

---

[1] Our definition of a diamond differs from related work, since the 4 sides of the polygon (when projected to each dimension) may differ in length, so the polygon may not be a rhombus, but in practice, the polygon does resemble a diamond in most cases.

velocities is different in our case. As an example, Figure 13.2(a) shows for one dimension $d \in D$, positions of state θ_i^o at time t_i^o and $o(t_{i+1}^o)$ at time t_{i+1}^o. The diamond formed by the velocity bounds $v_d^{\leqslant}, v_d^{\leqslant}, v_d^{\geqslant}$ and $v_d^{\geqslant}$ conservatively approximates the possible $(location, time)$ pairs. Note that it is possible to use a minimal bounding rectangle $\Box(o, t_i^o, t_{i+1}^o)$ instead of the diamond $\diamond(o, t_i^o, t_{i+1}^o)$ to conservatively approximate the $(location, time)$ space $S_{i,i+1}$. In cases, however, where the movement of an object in one dimension is biased in one direction, a rectangle may yield a very bad approximation (see Figure 13.2(b) for an example). Our index employs both approximations $\Box(o, t_i^o, t_{i+1}^o)$ and $\diamond(o, t_i^o, t_{i+1}^o)$ for spatio-temporal pruning; $\Box(o, t_i^o, t_{i+1}^o)$ is used for high-level indexing and filtering, while $\diamond(o, t_i^o, t_{i+1}^o)$ is used as a second-level filter.

13.3.2 Spatio-Temporal Filter

Based on the spatio-temporal approximation of an uncertain object as described in the previous section, it is possible to perform filtering during query processing. In the case of a $PST\tau\exists Q$ query, if none of the diamonds assigned to an object $o \in \mathcal{D}$ intersects the query window, then o is safely pruned. In turn, if a diamond of o is inside the query window $S^\Box$ in space, i.e. fully covered by $S^\Box$, at any point of time $t \in T^\Box$, then o is a *true hit* and, thus, o can be immediately reported as result of the query. Similarly, for a $PST\tau\forall Q$ query, objects whose diamonds do not intersect the query window during the *entire* query time-range $T^\Box$ can be pruned. On the other hand, an object o with a diamond which is fully covered by $S^\Box$ for each point of time $t \in T^\Box$ is immediately reported as a *true hit*. In order to employ the above spatio-temporal pruning conditions, for a diamond $\diamond(o, t_i^o, t_{i+1}^o)$ of an object o we need to determine the points of time when it intersects the query window $S^\Box$ in space, as well as the points of time when $\diamond(o, t_i^o, t_{i+1}^o)$ is fully covered by $S^\Box$. For this purpose, it is helpful to focus on the spatial domain $\mathcal{S}$ and interpret a diamond as well as the query as a time-parameterized (moving) rectangle. By doing so, we can adapt the techniques proposed in [137]: In general, a rectangle R_1 intersects (covers) another rectangle R_2, if and only if R_1 intersects (covers) R_2 in each dimension. Thus, for each spatial dimension d ($d \in \{1, \ldots D\}$), we compute the points of time when the extents of the rectangles intersect in that dimension and the points of time when the extents of the diamond rectangle are fully covered by the query rectangle $S^\Box$.

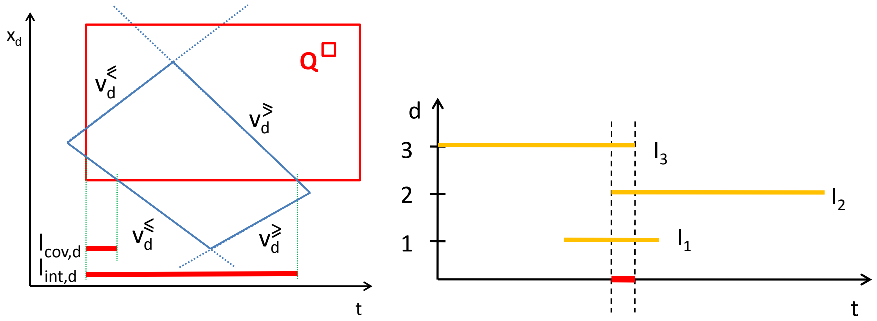
For a single dimension d, with Equation 13.1 the query window given by $Q_d^\Box$ intersects the diamond given by $o(t_i^o)[d]$, $o(t_{i+1}^o)[d]$, $v_d^{\leqslant}$, $v_d^{\leqslant}$, $v_d^{\geqslant}$ and $v_d^{\geqslant}$ within the points of time

$$I_{int,d} := \{t \in (T^\Box \cap [t_i^o, t_{i+1}^o]|I_d(t) \cap S_d^\Box\}.$$

Similarly, $Q_d^\Box$ fully covers the diamond within the points of time

$$I_{cov,d} := \{t \in T^\Box \cap [t_i^o, t_{i+1}^o]|I_d(t) \subseteq S_d^\Box\}.$$

An example is illustrated in Figure 13.3(a). To compute both sets $I_{int,d}$ and $I_{cov,d}$, we intersect the margins of the diamond with the query window resulting in a set of time intervals, which subsequently have to be intersected accordingly in order to derive $I_{int,d}$



(a) query intersection (b) dimension-wise intersections

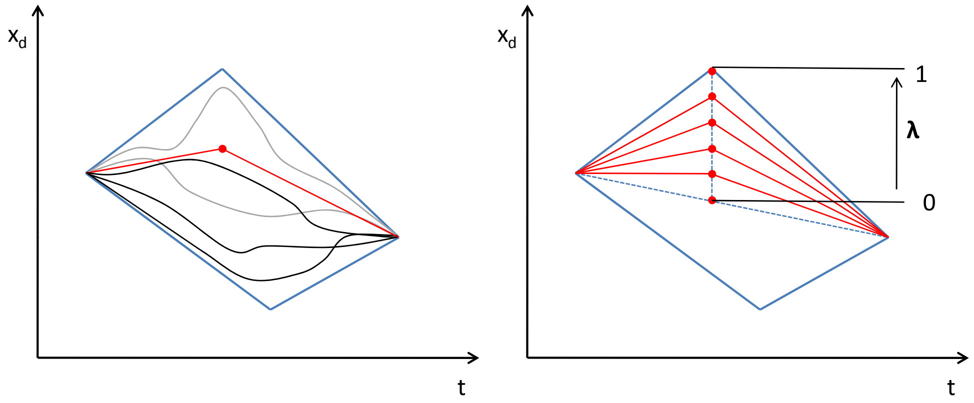
Figure 13.3: Intersection between query and diamond

and $I_{cov,d}$. Now, let us consider the overall intersection time interval $I_{int} = \bigcap_{d=1}^{D} I_{int,d}$ (e.g., see Figure 13.3(b)) and the overall points of covering time $I_{cov} = \bigcap_{d=1}^{D} I_{cov,d}$. In the case of a $PST\tau\exists Q$, if for an object $o \in \mathcal{D}$ there is no diamond yielding a non-empty set I_{int}, o can be safely pruned. If any diamond of o yields a non-empty set I_{cov}, o can be reported as result. In the case of a $PST\tau\forall Q$, an object $o \in \mathcal{D}$ can be safely rejected if all I_{int} of all diamonds of o together do not completely cover the query time range $T^{\Box}$. Contrary, if all I_{cov} of all diamonds of o together completely cover $T^{\Box}$, o can be reported as a result of $PST\tau\forall Q$.

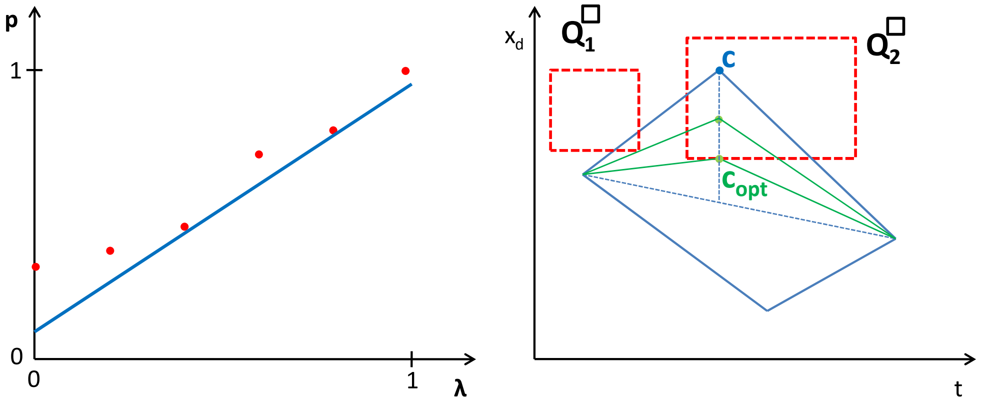
In summary, the spatio-temporal filter can be used to identify uncertain object trajectories having a probability of 100% or 0% intersecting (remaining in) the query region $Q^{\Box}$. Still, the probability threshold τ of the query is not considered by this filter. In addition, the object approximation may cover a lot of dead space if there exist outlier state-time pairs which determine one or more of the velocities, despite having a very low probability. In the following, we show how to exclude such unlikely outliers in order to shrink the approximation, while maintaining probabilistic guarantees that employ the probability threshold τ.

13.3.3 Probabilistic UST-Object Approximation

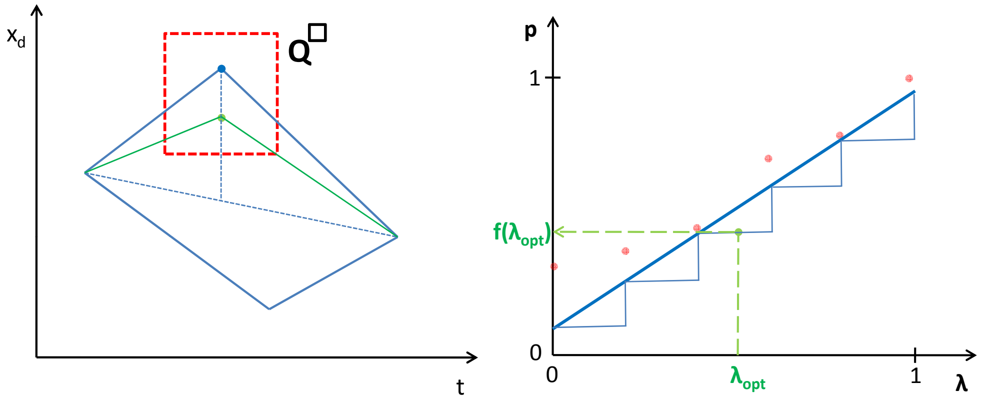
We now propose a tighter approximation, based on the intuition that the set of possible paths within a diamond is generally not uniformly distributed: paths that are close to the direct connection between the observed locations often are more likely than extreme paths along the edges of the diamond. Therefore, given a query with threshold τ, we can take advantage of a tighter approximation, which bounds all paths with cumulative probabilities τ to perform more effective pruning. Based on this idea, we exploit the Markov-chain model in order to compute new diamonds, which are spatio-temporal subregions, called subdiamonds, of the (full) diamond $\diamond(o, t_i^o, t_{i+1}^o)$, as depicted in Figure 13.4(a). For each such subdiamond, we will then show how to compute the cumulative probability of all possible trajectories of o passing only through this subdiamond. Let us focus on restricting the diamond in one direction of one dimension; we choose one dimension $d \in D$, and one



(a) A sub-diamond (b) Possible sub-diamonds



(c) linear optimization (d) Computing λ_{opt} ($\exists$-case)



(e) Computing λ_{opt} ($\forall$-case) (f) conservative approx.

Figure 13.4: Probabilistic diamonds

direction $dir \in \{\wedge, \vee\}$. Direction $\wedge$ ($\vee$) corresponds to the two diamond sides $v_d^{\leqslant}$ and $v_d^{\geqslant}$ ($v_d^{\leqslant}$ and $v_d^{\geqslant}$). To obtain the subdiamond, we scale the corresponding sides by a factor $\lambda \in [0,1]$ relative to the average velocity $v_d^{avg} = \frac{\theta_{i+1}^o[d] - \theta_i^o[d]}{t_{i+1}^o - t_i^o}$. We obtain the adjusted velocity values for direction $\wedge$ as follows:

$$\begin{aligned} v^{\leqslant\lambda}_d &= ((v_d^{\leqslant} - v_d^{avg}) \cdot \lambda) + v_d^{avg} \\ &= v_d^{\leqslant} \cdot \lambda + v_d^{avg} \cdot (1 - \lambda) \end{aligned}$$

and

$$\begin{aligned} v^{\geqslant\lambda}_d &= ((v_d^{\geqslant} - v_d^{avg}) \cdot \lambda) + v_d^{avg} \\ &= v_d^{\geqslant} \cdot \lambda + v_d^{avg} \cdot (1 - \lambda) \end{aligned}$$

The adjusted velocity values for direction $\vee$ can be computed analogously. Thus, for a given diamond $\diamond(o, t_i^o, t_{i+1}^o)$, dimension $d \in D$, direction $dir \in \{\wedge, \vee\}$ and scalar $\lambda \in [0,1]$, we obtain a smaller diamond $\diamond(o, t_i^o, t_{i+1}^o, d, dir, \lambda)$, derived from $\diamond(o, t_i^o, t_{i+1}^o)$ by scaling direction dir in dimension d by a factor of λ. Figure 13.4(b) illustrates some subdiamonds for one dimension, the $\wedge$ direction and for various values of λ.

To use such subdiamonds for probabilistic pruning, we first need to compute the probability $P(inside(o, \diamond(o, t_i^o, t_{i+1}^o, d, dir, \lambda)))$ that object o will remain within $\diamond(o, t_i^o, t_{i+1}^o, d, dir, \lambda)$ for the whole time interval $[t_i^o, t_{i+1}^o]$, in a correct and efficient way. The main challenge for correctness is to cope with temporal dependencies, i.e. the fact that the random variables $o(t_i)$ and $o(t_i + \delta t)$ are highly correlated. Thus, we cannot simply treat all random variables $o(t)$ as mutually independent and aggregate their individual distributions. To illustrate this issue, consider Figure 13.4(a), where one subdiamond is depicted. Assume that each of the five possible trajectories has a probability of 0.2. We can see that three trajectories are completely contained in the subdiamond, so that the probability $P(inside(o, \diamond(o, t_i^o, t_{i+1}^o, d, dir, \lambda)))$ that o fully remains in the subdiamond $\diamond(o, t_i^o, t_{i+1}^o, d, dir, \lambda)$ is 60%. However, multiplying for all time instants $t \in [t_i^o, t_{i+1}^o]$ the individual probabilities that o is located in $\diamond(o, t_i^o, t_{i+1}^o, d, dir, \lambda)$ at time t produces an arbitrarily small and incorrect result, as location dependencies (cf Section 11.3) are ignored.

Furthermore, due to the generally exponential number of possible trajectories, it is too expensive to compute $P(inside(o, \diamond(o, t_i^o, t_{i+1}^o, d, dir, \lambda)))$ by iterating over all possible trajectories. Instead, we compute this probability efficiently and correctly, as follows. To compute the probability of possible trajectories between θ_i^o at t_i^o and θ_{i+1}^o at t_{i+1}^o that are completely contained in $\diamond(o, t_i^o, t_{i+1}^o, d, dir, \lambda)$, an intuitive approach is to start at θ_i^o at time t_i^o and perform $t_{i+1}^o - t_i^o$ transitions using the Markov-chain $M^o(t)$. After each transition, we identify states that are outside $\diamond(o, t_i^o, t_{i+1}^o, d, dir, \lambda)$. Any possible trajectory which reaches such a state is flagged. Upon reaching t_{i+1}^o, we only need to consider possible trajectories in state θ_{i+1}^o, since all other worlds have become impossible due to the observation of o at t_{i+1}^o. The fraction of un-flagged worlds at state θ_{i+1}^o at time t_{i+1}^o yields the probability that o does not completely remain in $\diamond(o, t_i^o, t_{i+1}^o, d, dir, \lambda)$.

To formalize the above approach, we first rewrite the probability $P(inside(o, \diamond)|\Theta_i^o, \Theta_{i+1}^o))$ that the trajectory of o remains in the subdiamond $\diamond$,[2] given the observations Θ_i^o, Θ_{i+1}^o, using the definition of conditional probability:

$$P(inside(o, \diamond)|\Theta_i^o, \Theta_{i+1}^o) = \frac{P(o(t_{i+1}^o) = \theta_{i+1}^o|inside(o, \diamond), \Theta_i^o)}{P(o(t_{i+1}^o) = \theta_{i+1}^o|\Theta_i^o)},$$

where $P(o(t_{i+1}^o) = \theta_{i+1}^o|inside(o, \diamond), \Theta_i^o)$ denotes the probability that o reaches the state θ_{i+1}^o observed by observation Θ_{i+1}^o, given that o, starting at state θ_i^o at time t_i^o remains inside $\diamond$. $P(o(t_{i+1}^o) = \theta_{i+1}^o|\Theta_i^o)$ denotes the probability that state θ_{i+1}^o at time t_{i+1}^o is reached, given that o starts at state θ_i^o at time t_i^o, regardless whether o remains in $\diamond$.

To compute the latter two probabilities, we proceed as follows. Instead of a single probability vector $P(o, t)$ of length $|S|$, describing the state distribution of o at time t, we use two probability vectors $P(o, t)^+$ and $P(o, t)^-$, each of length $|S|$. An entry $P_s(o, t)^+$ corresponds to the joint probability of the event that o is located at state s at time t and the event $inside(o, \diamond, t)$ that o has (so far) been completely contained in the diamond in the time interval $[t_i^o, t_{i+1}^o]$. Analogously, an entry $P_s(o, t)^-$ corresponds to the probability that o is located at state s at time t and has already left the diamond at, or before, time t. Thus, vectors $P(o, t)^+$ and $P(o, t)^-$ describe the probability density function of o on the two dimensional event space $(state \times inside(o, \diamond, t))$, where the random variable $inside(o, \diamond, t)$ is dichotomous (i.e. true or false). Then, we proceed as follows: Starting at t_i^o, we perform $t_{i+1}^o - t_i^o$ transitions using alternated Markov-chains, using both vectors $P(o, t)^+$ and $P(o, t)^-$. Initially, $P(o, t_i^o)^+$ is a unit vector, having $P_{\theta_i^o}(o, t_i^o)^+ = 1$ (and all other entries set to zero) and $P_{\theta_i^o}(o, t_i^o)^-$ is the zero vector. Semantically, this means that at time t_i^o, o must be in state θ_i^o with a probability of one and must so far have retained to $\diamond$. At each transition from t_k to t_{k+1}, we decide for each reachable state $s \in S$, whether s is located in $\diamond$ at time t_{k+1}. This decision can be easily made by simply testing whether the coordinate $s[d]$ of state s in dimension d falls into interval of $\diamond$ using Equation 13.1. In the following, let $in(s, t, \diamond)$ be an indicator function that returns 1 if the state/time pair (s, t) is inside $\diamond$, and 0 otherwise. For each state s where $in(s, t_{k+1}, \diamond) = 0$, we add the probability of $P_s(o, t)^+$ to $P_s(o, t)^+$ and set $P_s(o, t)^+$ to zero. That is, in each iteration we compute

$$P(o, t_k + 1)^+ = P(o, t_k)^+ \cdot M^o(t), P(o, t_k + 1)^- = P(o, t_k)^- \cdot M^o(t)$$

and then for each state $1 \leq s \leq |S|$

$$P_s(o, t_k + 1)^- = P_s(o, t_k + 1)^- + P_(o, t_k + 1)^+ \cdot (1 - in(s, t, \diamond)),$$

$$P_s(o, t_k + 1)^+ = P_s(o, t_k + 1)^+ \cdot in(s, t, \diamond)$$

At the final time t_{i+1}^o, value $P(o, t_{i+1}^o)^+$ corresponds to $P(o(t_{i+1}^o) = \theta_{i+1}^o|inside(o, \diamond))$; i.e., the probability of o having reached the observed state θ_{i+1}^o at time t_{i+1}^o, without having left

---

[2] Since the context is clear, we simply use $\diamond$ to denote $\diamond(o, t_i^o, t_{i+1}^o, d, dir, \lambda)$.

$\diamond$, and the sum of $P(o, t^o_{i+1})^+$ and $P(o, t^o_{i+1})^-$ corresponds to the probability $P(o(t^o_{i+1}) = \theta^o_{i+1})$ that θ^o_{i+1} is reached at all. Thus:

$$P(inside(o, \diamond)) = \frac{P_{\theta^o_{i+1}}(o, t^o_{i+1})^+}{P_{\theta^o_{i+1}}(o, t^o_{i+1})^+ + P_{\theta^o_{i+1}}(o, t^o_{i+1})^-}$$

The overall time for computing the probability of a subdiamond $\diamond$ is in $O((t^o_{i+1} - t^o_i) \cdot |S|)$, since in each of the $t^o_{i+1} - t^o_i$ iterations, we only need to consider a finite number of $2|S|$ states. Since M is a sparse matrix, the vectors $P(o, t)^+$ and $P(o, t)^-$ remain sparse as well. This observation allows to further accelerate the computation of $P(inside(o, \diamond))$ using sparse matrix operations. Also note, that sampling methods are of limited use for computing $P(inside(o, \diamond)$. In addition to the approximative nature of sampling, most sampled paths derived by starting at θ^o_i at time t^o_i using the Markov chain $M^o(t)$, will not reach state θ^o_{i+1} at time t^o_{i+1}. Thus, a very large sample of paths has to be sampled, in order to find a representative number of paths that intersect θ^o_{i+1} at time t^o_{i+1}.

13.3.4 Finding the optimal Probabilistic Diamond

In the previous section, we described how to compute the probability of a probabilistic diamond $\diamond(o, t^o_i, t^o_{i+1}, d, dir, \lambda)$ from a diamond $\diamond(o, t^o_i, t^o_{i+1})$, dimension d, direction dir, and scaling factor λ. In this section we will show how to find, for a given query window $Q^\square$ and a given query predicate the subdiamond with the highest pruning power. Let us focus on PST$\tau\exists$ queries first. That is, our aim is to find a value for d, dir and λ, such that the resulting subdiamond $\diamond(o, t^o_i, t^o_{i+1}, d, dir, \lambda)$ does not intersect $Q^\square$, and at the same time it has a high probability $P(inside(o, \diamond))$. This probability can be used to prune o as we will show later. To allow effective pruning, we will show how to find, for a given query window $Q^\square$, diamonds having a sufficiently small probability of intersecting $Q^\square$ in order to prune these. Therefore, we will show how to find the best combination of parameters $d \in D, dir \in \{\vee, \wedge\}, \lambda \in [0, 1]$ such that the corresponding probabilistic diamond does not intersect $Q^\square$ and the probability $P(inside(o, \diamond(o, t^o_i, t^o_{i+1}, d, dir, \lambda)))$ is maximized. Formally, we want to efficiently determine

$$argmax_{d \in D, dir \in \{\vee, \wedge\}, \lambda \in [0,1]}[P(inside(o, \diamond(o, t^o_i, t^o_{i+1}, d, dir, \lambda))]$$

constrained to $Q^\square \cap \diamond(o, t^o_i, t^o_{i+1}, d, dir, \lambda) = \emptyset$.

For a single dimension d, and the *north* direction, a possible situation is depicted in Figure 13.4(d). Here, the projection $\diamond_d(o, t^o_i, t^o_{i+1})$ of the full diamond $\diamond(o, t^o_i, t^o_{i+1})$ to the d-th dimension and the projections $Q^\square_1[d]$ and $Q^\square_2[d]$ of two query windows $Q^\square_1$ and $Q^\square_2$ are depicted. The aim is to find the largest values λ_{opt} of λ, such that the corresponding probabilistic diamond $\diamond(o, t^o_i, t^o_{i+1}, d, \wedge, \lambda^\exists_{opt})$ which we call optimal subdiamond, does not intersect $Q^\square_1$ ($Q^\square_2$). To solve this problem, we distinguish between the following cases.

Case 1: the direct line between observations $< \theta^o_i, t^o_i >$ and $< \theta^o_{i+1}, t^o_{i+1} >$ in dimension d intersects $Q^\square[d]$. In this case there exists no probabilistic diamond for dimension d

in direction $\wedge$. If this is the case for each dimension, i.e. if the direct line between $< \theta_i^o, t_i^o >$ and $< \theta_{i+1}^o, t_{i+1}^o >$ intersects $Q^\square$ in the full space, then we cannot find any useful probabilistic subdiamond. Intuitively, for such a diamond the true probability of intersecting $Q^\square$ (after refinement), is expected to be very large, thus $\diamond(o, t_i^o, t_{i+1}^o)$ is unlikely to subjectable to probabilistic pruning, and thus we skip it entirely in the probabilistic pruning step of our algorithm.

Case 2: the direct line between $< \theta_i^o, t_i^o >$ and $< \theta_{i+1}^o, t_{i+1}^o >$ does not intersect $Q^\square[d]$, and we assume without loss of generality that $Q^\square[d]$ is located above this line.[3] In addition, in this case, the time value of the north corner c of $\diamond_d(o, t_i^o, t_{i+1}^o)$ is located in the interval $T^\square$ (e.g., see $Q_2^\square$ in Figure 13.4(d)).[4] In this case, the edge $v_{opt}^{\leqslant}$ of the optimal subdiamond $\diamond(o, t_i^o, t_{i+1}^o, d, dim, \lambda_{opt}^\exists)$ is given by $< \theta_i^o, t_i^o >$ and (s, t) where s corresponds to the lower bound of $S^\square[d]$ and t equals to the time component of c.

Case 3: $Q^\square$ is above the direct line between $< \theta_i^o, t_i^o >$ and $< \theta_{i+1}^o, t_{i+1}^o >$ (as in Case 2), but the time value of the north corner c of $\diamond_d(o, t_i^o, t_{i+1}^o)$ is not located in the time interval $T^\square$ (e.g. $Q_1^\square$ of Figure 13.4(d)). In this case, the optimal subdiamond must touch a corner of $Q^\square[d]$ due to convexity of both $Q^\square[d]$ and any diamond. If $Q^\square[d]$ is located to the left of c (the right direction is handled symmetrically), then the edge $v_{opt}^{\leqslant}$ of the optimal subdiamond is given by the line between $< \theta_i^o, t_i^o >$ and the lower right corner of $Q^\square[d]$ (e.g., see Figure 13.4(d)).

The optimal value $\lambda_{opt}^\exists$ for cases 2 and 3 equals the quotient $\frac{v_{opt}^{\leqslant} - v_{avg}}{v^{\leqslant} - v_{avg}}$, i.e., the fraction of the maximum velocity of the optimal subdiamond and the maximum velocity of the full diamond, both normalized by the average velocity $v_{avg} = \frac{\theta_{i+1}^o[d] - \theta_i^o[d]}{t_{i+1}^o - t_i^o}$. After identifying the value for $\lambda_{opt}^\exists$, for a dimension d and a direction dir, we can compute the probability of the corresponding subdiamond $\diamond(o, t_i^o, t_{i+1}^o, d, dir, \lambda_{opt}^\exists)$. Since we can guarantee, that any path in this subdiamond does not intersect the query window, we can obtain a lower bound

$$P_{LB}(never(o, t_i^o, t_{i+1}^o, Q^\square)) = P(inside(o, \diamond(o, t_i^o, t_{i+1}^o, d, dir, \lambda_{opt}^\exists))) \tag{13.2}$$

of the event that o never intersects the query window in the time interval $[t_i^o, t_{i+1}^o]$. This directly yields an upper bound

$$P_{UB}(sometimes(o, t_i^o, t_{i+1}^o, Q^\square)) = 1 - P(inside(o, \diamond(o, t_i^o, t_{i+1}^o, d, dir, \lambda_{opt}^\exists))) \tag{13.3}$$

of the probability that the reverse event that o intersects the query window at least once in $[t_i^o, t_{i+1}^o]$. This bound can be used for probabilistic pruning for PST$\tau\exists$ queries, as we will see in Section 13.3.6.

For PST$\tau\forall$ queries we can naively use the probabilistic subdiamond computed above, exploiting the fact that any world that does not qualify a PST$\tau\exists$ query $Q^\square$, cannot satisfy the corresponding PST$\tau\forall$ query $Q^\square$. Still, we can do better: We can extend the probabilistic subdiamond (i.e., increase the value of λ), until it becomes possible that a path in

---

[3] If $Q^\square[d]$ is below the line, we consider direction $dir = \vee$ symmetrically.

[4] Corner c is given by the intersection of lines $(o(t_i^o), t_i^o) + v^{\leqslant}$ and $(o(t_{i+1}^o), t_{i+1}^o) + v^{\geqslant}$.

the subdiamond remains completely in $Q^{\square}$. In Case 1, where v_{exp} intersects $Q^{\square}$, again we can find no probabilistic subdiamond; in both other cases, we find the best probabilistic diamond for each dimension and each direction, as the maximum $\lambda^{\forall}_{opt}$, such that the resulting subdiamond does not contain both corners of $Q^{\square}[d]$ facing $\diamond(o, t^o_i, t^o_{i+1})$. An example is given in Figure 13.4(e). Here, $\lambda^{\forall}_{opt}$ is chosen for the north direction, such that $v^{\geqslant}$ meets the lower right corner of $Q^{\square}[d]$. The resulting diamond $\diamond(o, t^o_i, t^o_{i+1}, d, dir, \lambda^{\forall}_{opt})$ is guaranteed to not contain any path that satisfies a PST$\tau\forall$ query. The reason is that $\lambda^{\forall}_{opt}$ is chosen such that there is guaranteed to be one point of time [5] when the query window $Q^{\square}$ is not intersected by $\diamond(o, t^o_i, t^o_{i+1}, d, dir, \lambda^{\forall}_{opt})$.

This observation allows to derive the lower bound probability

$$P_{LB}(sometimes\_not(o, t^o_i, t^o_{i+1}, Q^{\square})) = P(inside(o, \diamond(o, t^o_i, t^o_{i+1}, d, dir, \lambda^{\forall}_{opt})))$$

of the event that o misses the query window at least once. Again, we derive an upper bound probability

$$P_{UB}(always(o, t^o_i, t^o_{i+1}, Q^{\square})) = P(inside(o, \diamond(o, t^o_i, t^o_{i+1}, d, dir, \lambda^{\forall}_{opt})))$$

of the reverse event that o is always located in $Q^{\square}$, which can be used for pruning for PST$\tau\exists$ queries (see Section 13.3.6).

13.3.5 Approximating Probabilistic Diamonds

The main goal of our index structure, proposed in Section 13.4, is to avoid expensive probability computations for subdiamonds. Since the query window is not known in advance, $2D$ computations (i.e., one for each dimension and direction) have to performed in order to identify the optimal subdiamond for a given query and candidate object o. To avoid these computations at run-time, we propose to precompute, for each diamond $\diamond(o, t^o_i, t^o_{i+1})$ in $\mathcal{D}$, probabilistic subdiamonds for each dimension and direction and for a set Λ of λ-values. This yields a catalogue of probability values, i.e. a probability for each $\diamond(o, t^o_i, t^o_{i+1}, d, dir, \lambda), d \in D, dir \in \{\vee, \wedge\}, \lambda \in \Lambda$.

Given a query window, the optimal value λ_{opt} computed in Section 13.3.4 may not be in Λ. Thus, we need to conservatively approximate the probability of probabilistic diamonds $\diamond(o, t^o_i, t^o_{i+1}, d, dir, \lambda)$ for which $\lambda \notin \Lambda$. We propose to use a conservative linear approximation of $P(inside(o, \diamond(o, t^o_i, t^o_{i+1}, d, dir, \lambda)))$ which increases monotonically with λ, using the precomputed probability values. For example, Figure 13.4(c), shows the $(\lambda, probability)$-space, for six values $\Lambda = \{0, 0.2, 0.4, 0.6, 0.8, 1\}$. The corresponding precomputed pairs $(\lambda, P(inside(o, \diamond(o, t^o_i, t^o_{i+1}, d, dir, \lambda))))$ are depicted. Our goal is to find a function $f(\lambda)$ that minimizes the error with respect to $P(inside(o, \diamond(o, t^o_i, t^o_{i+1}, d, dir, \lambda))$, while ensuring that $\forall \lambda \in [0, 1] : f(\lambda) \leq P(inside(o, \diamond(o, t^o_i, t^o_{i+1}, d, dir, \lambda))$. The latter constraint is required to maintain the conservativeness property of the approximation, which will be required for pruning. We model this as a linear programming problem: find a linear function

---

[5] corresponding to the corner which is not intersected by $\diamond(o, t^o_i, t^o_{i+1}, d, dir, \lambda^{\forall}_{opt})$

$l(\lambda) = a \cdot \lambda + b$ that minimizes the aggregate error with respect to the sample points, under the constraint that the approximation line does not exceed any of the sample values (e.g., the line in Figure 13.4(c)). That is, we compute:

$$minf(a,b), \text{ where } f(a,b) = \sum_{\lambda \in \Lambda} P(\lambda) - (a + b \cdot \lambda)$$

$$\text{subject to: } \forall \lambda \in \Lambda : P(\lambda) \geq a + b \cdot \lambda$$

Using the simplex algorithm we are able to solve this optimization problem extremely fast. In summary, a probabilistic spatio-temporal object o is approximated by a set of $|\Theta^o| - 1$ diamonds, one for each subsequent time points $t_i^o, t_{i+1}^o \in \Theta^o$. Each diamond approximation contains its spatio-temporal diamond $\diamond(o, t_i^o, t_{i+1}^o)$, consisting of four real values $v^{\leqslant}, v^{\preccurlyeq}, v^{\geqslant}, v^{\succcurlyeq}$, and a set of $2 \cdot D$ linear approximation functions $f_{d,dir}(\lambda)$, one for each dimension $d \in D$ and each direction $dir \in \{\vee, \wedge\}$. Next, we will show how to use these object approximations for efficient query processing over uncertain spatio-temporal data.

13.3.6 Probabilistic Filter

For each dimension $d \in D$ and direction $dir \in \{\vee, \wedge\}$, we now have a linear function to approximate all $(\lambda, P(o, t_i^o, t_{i+1}^o, d, dim, \lambda))$. However, using this line directly may violate the conservativeness property, since the true function may have any monotonic increasing form, and thus, for a value λ_Q located in between two values λ_1 and λ_2 ($\lambda_1, \lambda_2 \in \Lambda, \lambda_1 < \lambda_Q < \lambda_2$) the probability is bounded by $P(\lambda_1) \leq P(\lambda_Q) \leq P(\lambda_2)$. To avoid this problem, we can exploit that the catalogue Λ is the same for all diamonds, dimensions and directions. Thus, we chose the function $f(\lambda) = l(\lfloor \lambda \rfloor)$, where $\lfloor \lambda \rfloor$ denotes the largest element of Λ such that $\lfloor \lambda \rfloor \leq \lambda$. In our running example, the function $f(\lambda)$ is depicted in Figure 13.4(f). In this example, assume that we have computed an optimal value λ_{opt} in the previous steps. The corresponding conservative approximation $f(\lambda_{opt})$ is shown.

Now, we show how these probability bounds can be used to bound the probability that an object (i.e. its corresponding chain of diamonds) satisfies the query predicate. This is done by probing each uncertain trajectory approximation (each necklace) on the query region $Q^{\Box}$. Obviously, we only have to take into account diamonds intersecting the query time range $T^{\Box}$. In turn, when probing an uncertain trajectory approximation $\diamond(o, t_i^o, t_{i+1}^o)$ on the query range $Q^{\Box}$, we only have to take the time range $[t_i^o, t_{i+1}^o]$ into account; i.e., if the time range $T^{\Box}$ of the query spans beyond $[t_i^o, t_{i+1}^o]$, we truncate $T^{\Box}$ accordingly. Consequently, in the case where more than one diamonds of an object intersect $T^{\Box}$, we can split $Q^{\Box}$ at the time dimension and separately probe the object diamonds on the corresponding query parts. The resulting probabilities obtained for individual diamonds can be treated as independent.

Lemma 13.1. Let $\diamond(o, t_i^o, t_{i+1}^o)$, $\diamond(o, t_{i+1}^o, t_k)$ be two successive diamonds of object o and $\diamond_1 := \diamond(o, t_i^o, t_{i+1}^o, d_1, dir_1, \lambda_1)$, $\diamond_2 := \diamond(o, t_{i+1}^o, t_k, d_2, dir_2, \lambda_2)$ be probabilistic subdiamonds, associated with respective probabilities $P(\diamond_1)$ and $P(\diamond_2)$ that o intersects these subdiamonds. Then, the probability $P(\diamond_1 \wedge \diamond_2)$ that o intersects both subdiamonds, is given by

$$P(inside(o, \diamond_1) \wedge inside(o, \diamond_2)) = P(inside(o, \diamond_1)) \cdot P(inside(o, \diamond_2))$$

Proof. We first rewrite $P(inside(o, \diamond_1) \wedge inside(o, \diamond_2))$ using conditional probabilities.

$$P(inside(o, \diamond_1) \wedge inside(o, \diamond_2)) = P(inside(o, \diamond_1)) \cdot P(inside(o, \diamond_2) | inside(o, \diamond_1))$$

Furthermore, we exploit the knowledge that object o is at the observed location $o(t^o_{i+1})$ at time t^o_{i+1}

$$P(inside(o, \diamond_1) \wedge inside(o, \diamond_2)) = P(inside(o, \diamond_1)) \cdot P(inside(o, \diamond_2) | inside(o, \diamond_1) \wedge o(t^o_{i+1}))$$

Based on the Markov model assumption, we know that, given the position at t^o_{i+1}, the behavior of o in the time interval $[t^o_{i+1}, t_k]$ is independent of any position at times $t < t^o_{i+1}$. Thus, we obtain:

$$P(\diamond_1 \wedge \diamond_2) = P(\diamond_1) \cdot P(\diamond_2 | o(t^o_{i+1}))$$

Finally, the lemma is proved based on the fact that the position $o(t^o_{i+1})$ has been observed, and thus, is not a random variable. □

Lemma 13.1 shows that the random events of two successive probabilistic diamonds of the same object are conditionally independent, given the observation in between them. This observation allows us to compute the probability $P^{\exists}(o)$ that the whole chain of diamonds of o intersects a query window $Q^{\square}$. Let $\{t^o_i, t^o_{i+1}\} \subseteq_{seq} \Theta^o$ be the set of pairs of subsequent observations in Θ^o. Then,

$$P^{\exists}(o) = P(\bigvee_{\{t^o_i, t^o_{i+1}\} \subseteq_{seq} \Theta^o} sometimes(o, t^o_i, t^o_{i+1}, Q^{\square}))$$

That is, o satisfies a PST$\tau\exists$ query, if and only if at least one diamond of o intersects $Q^{\square}$ at least once. Rewriting yields

$$P^{\exists}(o) = 1 - P(\bigwedge_{\{t^o_i, t^o_{i+1}\} \subseteq_{seq} \Theta^o} never(o, t^o_i, t^o_{i+1}, Q^{\square}))$$

Exploiting Lemma 13.1 yields

$$P^{\exists}(o) = 1 - \prod_{\{t^o_i, t^o_{i+1}\} \subseteq_{seq} \Theta^o} P(never(o, t^o_i, t^o_{i+1}, Q^{\square}))$$

Using our probability bounds derived in Section 13.3.4, we obtain

$$P^{\exists}(o) \leq 1 - \prod_{\{t^o_i, t^o_{i+1}\} \subseteq_{seq} \Theta^o} P_{LB}(never(o, t^o_i, t^o_{i+1}, Q^{\square}))$$

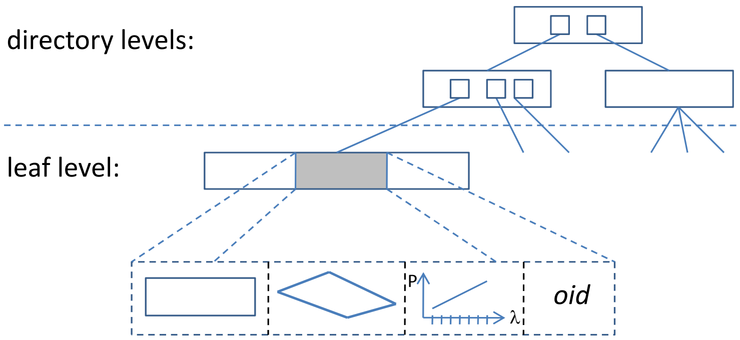


Figure 13.5: The UST-tree.

which can be used to prune o, if

$$1-\prod_{\{t_i^o,t_{i+1}^o\}\subseteq_{seq}\Theta^o} P_{LB}(never(o,t_i^o,t_{i+1}^o,Q^\square)) < \tau \tag{13.4}$$

Analogously, we can prune an object o for a $\mathrm{PST}\tau\forall$ query if

$$P^\exists(o)\leq 1-\prod_{\{t_i^o,t_{i+1}^o\}\subseteq_{seq}\Theta^o} P_{LB}(sometimes\_not(o,t_i^o,t_{i+1}^o,Q^\square))<\tau \tag{13.5}$$

If Equation 13.4 (Equation 13.5) cannot be applied for pruning, we propose to iteratively refine single diamonds of o.[6] Thus, the exact probability $P(sometimes(o,t_i^o,t_{i+1}^o,Q^\square))$ or $P(always(o,t_i^o,t_{i+1}^o,Q^\square))$ are computed using the technique proposed in the previous chapter, respectively. This exact probability of a single diamond can then be used to re-apply the pruning criterion of Equation 13.4 (Equation 13.5), by using the true probability as lower bound. When all diamonds of o have been refined, Equation 13.4 (Equation 13.5) yields the exact probability $P^\exists(o)$ or $P^\forall(o)$, respectively.

13.4 The UST-Tree

In the previous section, we showed that we can precompute a set of approximations for each object, which can be progressively used to prune an object during query evaluation. In this section, we introduce the UST-tree, which is an R-tree-based hierarchical index structure, designed to organize the object approximations and efficiently prune objects that may not possibly qualify the query; for the remaining objects the query is directly verified based on their Markov models, as described in Section 12.4 (*refinement step*). Section 13.4.1 describes the structure of the UST-tree and Section 13.4.2 presents a generic query processing algorithm for answering both $PST\tau\exists Q$ and $PST\tau\forall Q$ probabilistic query types efficiently.

---

[6] Heuristics to determine the order in which diamonds are refined are out of scope of this work.

13.4.1 Architecture

The UST-tree index is a hierarchical disk-based index. The basic structure is illustrated in Figure 13.5. An entry on the leaf level corresponds to an approximation of an object o represented by a quadruple $(\Box(o, t_i^o, t_{i+1}^o), \diamond(o, t_i^o, t_{i+1}^o), \{f_{d,dir} : d \in D, dir \in \{\vee, \wedge\}\}, oid)$, containing (i) the MBR approximation $\Box(o, t_i^o, t_{i+1}^o)$ (cf. Section 13.3.1), (ii) the diamond approximation $\diamond(o, t_i^o, t_{i+1}^o)$ (cf. Section 13.3.1), (iii) a set $\{f_{d,dir} : d \in D, dir \in \{\vee, \wedge\}\}$ of $2 \cdot D$ linear approximation functions for the precomputed probabilistic diamonds of o (cf. Section 13.3.5), and (iv) a pointer oid to the exact uncertain spatio-temporal object description (raw object data). Intermediate node entries of the UST-tree have exactly the same structure as in an R-tree; i.e., each entry contains a pointer referencing its child node and the MBR of all MBR approximations stored in pointed subtree. Note that the necklace of each object is decomposed into diamonds, which are stored independently in the leaf nodes of the tree. Since the directory structure of the UST-tree is identical to that of the R-tree, the UST-tree uses the same methods as the R*-tree [21] to handle updates. Update operations on the UST-tree are handled in the same way as in an R*-tree. Consequently, since the structure of intermediate nodes comply with that of the R*-tree, split and merge operations on intermediate nodes are quite obvious. For the leaf level, we also adopt the split and merge heuristics of the $R^* - tree$ by just taking the mbr-entries into consideration.

13.4.2 Query Evaluation

Given a spatio-temporal query window $Q^\Box$, the UST-tree is hierarchically traversed starting from the root, recursively visiting entries whose MBRs intersect $Q^\Box$; i.e., the subtree of an intermediate entry e is pruned if $e.mbr \cap Q^\Box = \emptyset$. For each leaf node entry e, we progressively use the spatio-temporal and probabilistic diamond approximations stored in e to filter the corresponding object.

In the spatio-temporal filter step, we first use $\Box(o, t_i^o, t_{i+1}^o)$ (ST-MBR Filter) using simple rectangle intersection tests. If this filter fails, we proceed using $\diamond(o, t_i^o, t_{i+1}^o)$ (ST-Diamond filter) by performing intersection tests against $Q^\Box$ as described in Section 13.3.2. Note that sometimes multiple leaf entries associated with an object are required to prune an object or confirm whether it is a *true hit*. Therefore, candidates are stored in a list until all their diamond approximations have been evaluated.

Finally, for the remaining candidates we exploit the probabilistic filter (Probabilistic Diamond Filter) as described in Section 13.3.6. Thereby, we use the linear approximation functions $\{f_{d,dir} : d \in D, dir \in \{\vee, \wedge\}\}$ stored in the leaf-node entry in order to derive an upper bound of the qualification probability $P(\exists t \in (T^\Box \cap [t_i^o, t_{i+1}^o]) : o(t) \in S^\Box)$ or $P(\forall t \in (T^\Box \cap [t_i^o, t_{i+1}^o]) : o(t) \in S^\Box)$ (depending on the query predicate). For each object o which is not pruned (or reported as true hit), we accumulate in a list $L(o)$ all upper bounds of its qualification probabilities from the leaf entries that index the diamonds of o. After collecting all candidate objects, the qualification probabilities stored in the list $L(o)$ for each candidate o are aggregated in order to derive the upper bound of the overall qualification probability $P(\exists t \in T^\Box : o(t) \in S^\Box)$ for o, as described in Section 13.3.6. If this probability falls below τ we can skip o, otherwise we have to refine o by accessing the exact object data referenced by *oid*.

13.5 Experimental Evaluation

13.5.1 Datasets and Experimental Setting

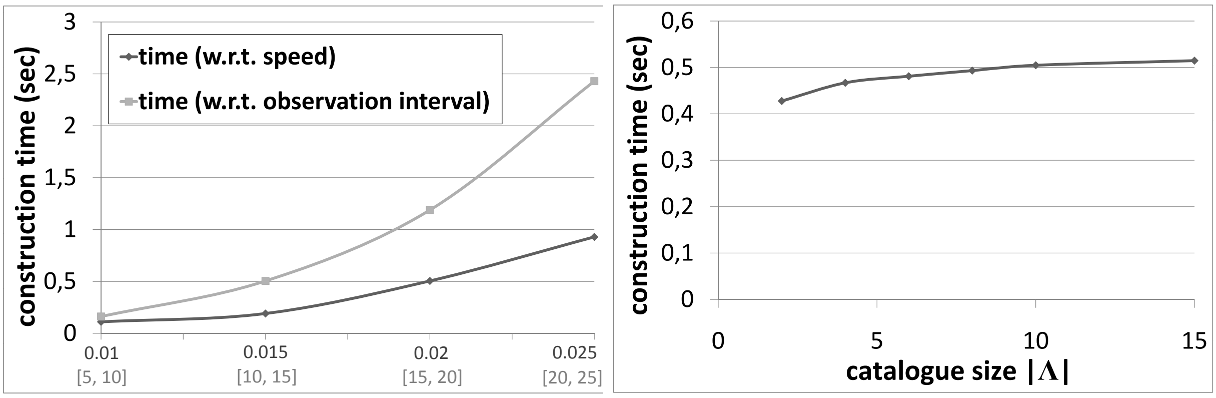
In order to evaluate the proposed techniques we used data derived from a real application and several synthetic data sets.

Real Data. As a basis for the real world data served the trajectory data set containing one-week trajectories of 10,357 taxis in Beijing from [174]. This heterogeneous dataset contains trajectories having different samples rates, ranging from one sample every fives seconds to one sample every 10 minutes. The average time between localization updates (observations) is 177 seconds. The average distance between two observations is 623 m. We applied the techniques from [48] to obtain both a set of possible states (mostly corresponding crossroads) and a transition matrix reflecting the possible movements of the taxis. We only included data of taxis where the time between two GPS signals (observations) is no more than two minutes to train the Markov chain. The resulting data set consists of 3008 states and 11699 possible transitions between theses states.

Synthetic Data. In order to demonstrate the behavior of the proposed techniques depending on the underlying data we also generated a set of synthetic data sets with different characteristics. For the possible states, we generated n points uniformly distributed in the $[0,1]^2$ space. Each point was then connected to the points which have an Euclidean distance smaller than ϵ. Those connections correspond to the possible movements of an object in the space and we randomly assigned probabilities to each connection such that the sum of all outgoing edges sums up to 1. These values are the entries for the corresponding transition matrix. As a default for this dataset we generated 1000 objects each with 100 observations ($=$ 99.000 probabilistic diamonds) and the parameters were set to $n = 10000, \epsilon = 0.02$ and the catalogue size $|\Lambda| = 10$.

Observations. Additionally to the positions of the states and the transition matrix, we further need observations from each object in order to build a database. The observations were constructed by a directed random walk through the underlying graph (states $=$ vertices and non-zero transitions $=$ edges). At some time steps we memorize the current position of the object and take these time-state-tuples as an observation of the object. The time steps between two successive observations was randomly chosen from the interval [10,15] if not stated otherwise. For the observations of the real dataset we used the GPS data of taxis, where the time between two signals is between 2 and 20 minutes.

All experiments were run on a Quad Intel Xeon server running Windows Server 2008 with 16 GB RAM and 3.0 GHz. The UST-tree was implemented in Java. For all operations involving matrix operations (e.g., the refinement step of queries) we used MATLAB for efficient processing. All query performance evaluation results are averaged over 1000 queries. The spatial extent of the query windows in each dimension was set to 0.1 and the



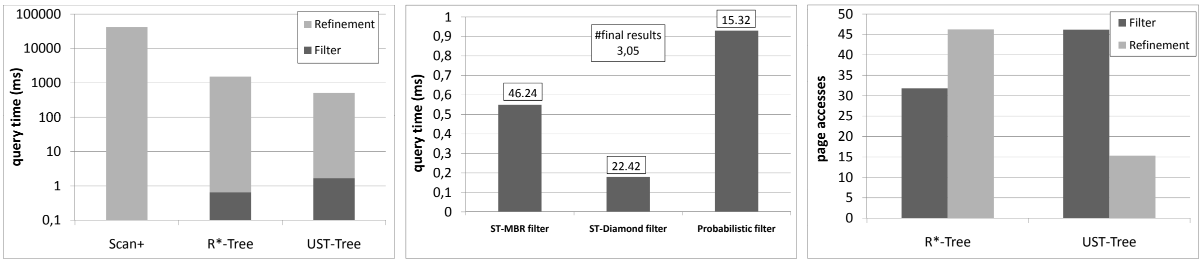
(a) diamond parameter (b) catalogue size

Figure 13.6: Diamond construction

duration of the queries was set to 10 time steps by default. Unless otherwise stated, we experimented with PST$\tau\exists$ queries, with $\tau = 0.5$. The page size of the tree was set to 4 KB. Our experiments assess the construction cost of the UST-tree structure and its performance on query evaluation. For experiments regarding the effectivity of the Markov-chain model, and an evaluation of its capability to capture the real world in various applications, we refer to previous work, e.g., [12, 48, 79, 135], where Markov Chains were proved successful in modeling spatio-temporal data.

13.5.2 UST-tree Construction

The first experiment investigates the cost of index construction. In particular, we evaluate the cost for generating the spatio-temporal and probabilistic diamond approximations used to build the entries of the leaf level; this is the bottleneck of constructing and updating the tree, since restructuring operations always take at most 1ms. On the other hand, constructing the probabilistic diamonds is typically 2-3 orders of magnitude costlier, as illustrated in Figure 13.6(a). Still, this cost is reasonable, since the construction of a probabilistic diamond is comparable to the construction of $2 \cdot D \cdot |\Lambda|$ subdiamonds, which in turn corresponds to one refinement step (considering the subdiamond as a query window). Construction times pay off, when the query load on the database is reasonable. Figure 13.6(a) illustrates the construction time as a function of the speed of the objects (upper x-axis values) and the number of time steps between successive observations (lower x-axis values). From a theoretical point of view, both parameters linearly increase the number of reachable states, i.e., the density of the sparse vectors representing the uncertain position of an object at one point of time. The results reflect the theoretical considerations showing a quadratic runtime behavior with respect to both parameters. In a streaming scenario with several updates/insertions per second and large probabilistic diamonds (due to high speed of objects or large intervals between observations), the construction of probabilistic diamonds can be performed in parallel and is therefore still feasible. Figure 13.6(b) shows



(a) CPU Comparison (b) CPU Time of filter (c) Page accesses

Figure 13.7: Overall performance (synthetic data set)

the construction cost as a function of parameter $|\Lambda|$ (which determines the number of subdiamonds). Theoretically this parameter should have a linear impact on the construction time. However our implementation exploits the monotonicity of the uncertain trajectories regarding probabilistic subdiamonds; a trajectory which is not included in the probability of a subdiamond, is also excluded from larger subdiamonds in the same dimension and direction. This explains the sublinear runtime w.r.t. $|\Lambda|$.

13.5.3 Query Performance

In the first set of query performance experiments, we compare the cost of using UST-tree with two competitors on synthetic data (see Figure 13.7). *Scan+* is a scan based query processing implementation, i.e., without employing any index [66]. For each pair of two successive observations of an object, refinement is performed immediately, i.e., there is no filter cost. We enhanced the implementation of *Scan+* by prepending a simple temporal filter, which only considers observation pairs which temporally overlap the query window. The *R*-Tree* competitor approximates all possible locations (i.e. state-time pairs) between two successive observations of an object using only $\Box(o, t_i, t_j)$. These MBRs are then indexed using a conventional R*-Tree [21]. In Figure 13.7(a), we show the average CPU cost per query (I/O cost is not the bottleneck in this problem), for the three competitors. The cost are split into filter and refinement costs. Although the R*-Tree has lower filter cost, the overall query performance of the UST-tree is around 3 times better than that of the R*-Tree (note the logarithmic scale). This is attributed to the effectiveness of the different filter steps used by the UST-tree; the overhead of the UST-tree filter is negligible compared to the savings in refinement cost.

Figure 13.7(b) shows the cost and the effectiveness of the individual filter steps of the filter-refinement pipeline used by the UST-tree. The bars show the overall runtime (query time) of each filter and the numbers on top of the bars show the effectiveness of the filter in terms of remaining (observation pair) candidates after the corresponding filter has been applied. We clearly see that the spatio-temporal filters reduce the number of candidates and, thus, the number of required refinements, drastically. We can also observe that the probabilistic filter can reduce the number of refinements by 30% after applying the sequence of spatio-temporal filters. Comparing the cost of the probabilistic filter (which is comparable to that of the spatio-temporal filter) to the cost of candidate refinement, we can observe that the cost required to perform the probabilistic filter can be neglected. This experiment shows that each of the filters incorporated in the UST-tree indeed pays off in terms of CPU cost.

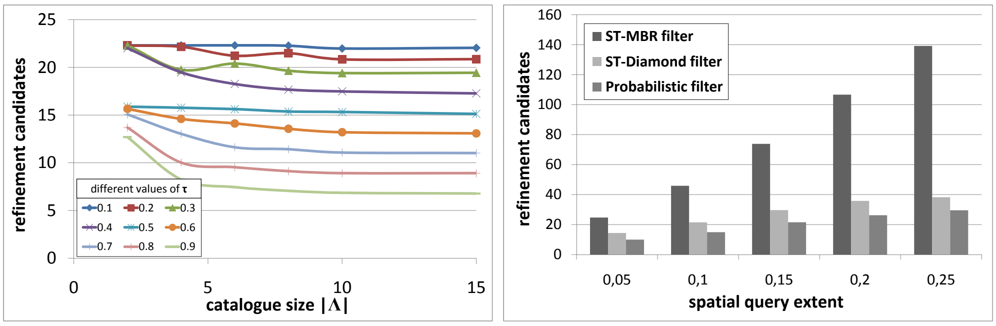
Although I/O cost is not the bottleneck under our setting, the I/O costs of *R*-Tree* and the UST-tree are illustrated in Figure 13.7(c) for completeness. Filter cost here means all costs which occur during the traversal of the corresponding index structure, i.e., access to intermediate and leaf nodes. Refinement cost includes the number of page accesses to refine the observation pairs that pass the filter step, assuming one I/O per such pair. Note, that the cost of a refinement can be much higher than one page access (e.g. if the Markov Chain, which can become very large does not fit in one disk page) under different settings. The UST-tree has higher filtering cost, since the representation of the probabilistic diamonds requires more space and the tree is larger than the R*-Tree, which only stores MBR approximations but incurs much higher I/O cost for refinements.

The above experiments unveil that the most costly operation is the refinement of spatio-temporal diamonds; thus, we now take a closer look at the effectiveness of the three different methods on pruning spatio-temporal diamonds. The next experiments measure the number of spatio-temporal diamonds which have to be refined at the refinement step; these results can be directly translated to runtime differences of the different approaches.

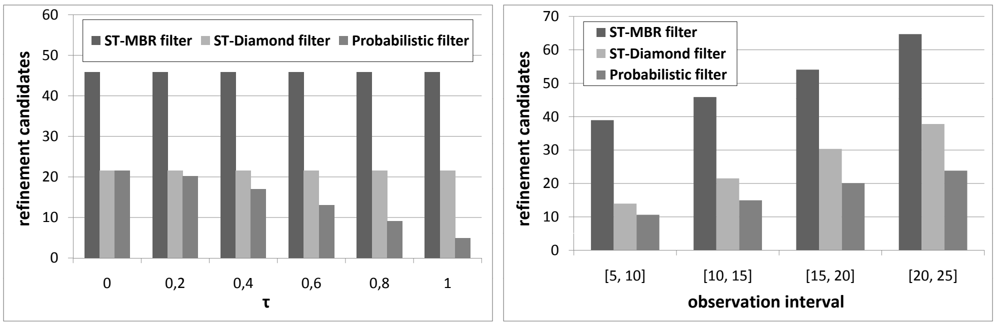
Size of the Catalogue $|\Lambda|$. An important tuning parameter for the index is the size of the catalogue which is used for building the probabilistic diamond approximations. In Figure 13.8(a), it can be observed that the filter effectiveness converges at around $|\Lambda| = 10$ (default value for the experiments). Depending on the query parameter τ, a too small catalogue yields up to twice as much candidates which have to be refined. Note that the number of refinement candidates does not decrease monotonically in $|\Lambda|$. In general a larger catalogue results in a linear function with a larger approximation error. However, the step-function for the conservative approximation becomes smoother which results in a smaller approximation error. Because of these two contrary effects a larger catalogue does not always result in higher filter effectiveness.

Query Parameters. The characteristics of the query have different implications on the index performance. Increasing the spatial extent of the query obviously yields more candidates since more diamonds in the database are affected (cf. Figure 13.8(b)). The spatio-temporal filter utilizing the diamond approximations becomes more effective in comparison to the ST-MBR-Filter. The percentage of the diamonds which can be pruned using the probabilistic filter remains rather constant (at around 30%) in comparison to the spatio-temporal filter. Another query parameter is the temporal extent of the query. Increasing the length of the query time window $T^{\Box}$ increases the number of refinement candidates. The results are very similar to the results when increasing the spatial extent of the query.

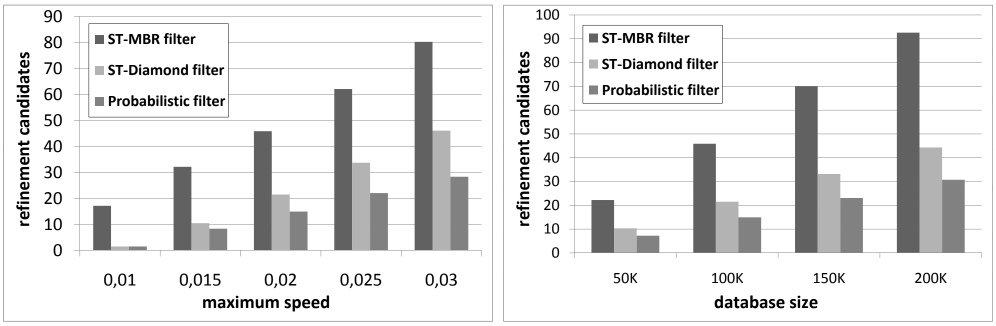
Changing the value of τ obviously only affects the probabilistic filter (cf. Figure 13.8(c)). The higher τ is set, the more candidates can be pruned by the probabilistic filter. From a value of around 20%, the candidates which have to be refined decrease linearly with τ.



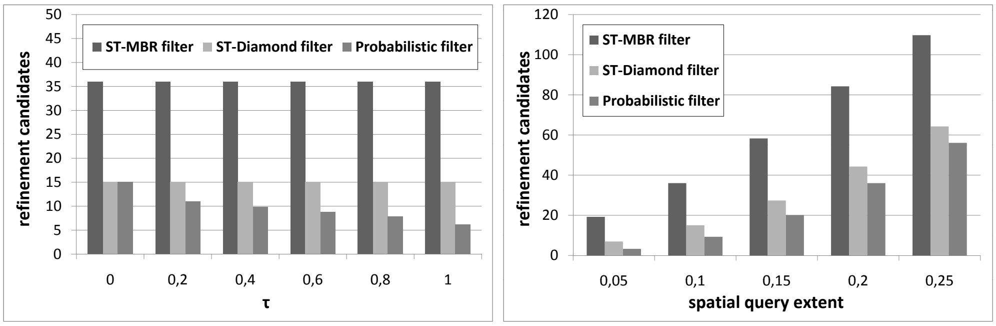
(a) |Λ| for different values of τ (b) query extent



(c) varying τ (d) observation time interval

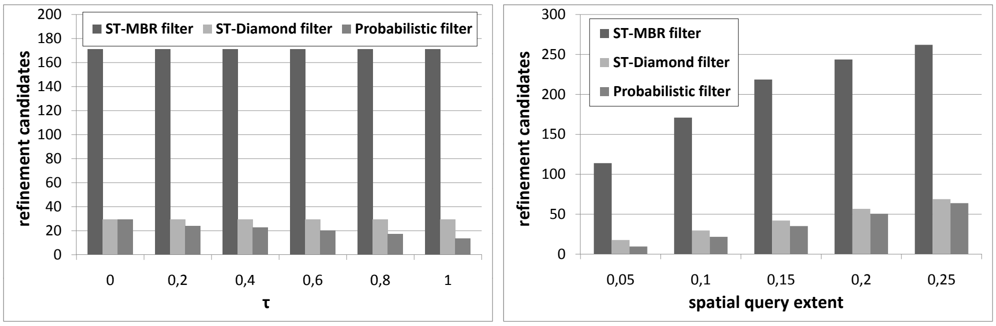


(e) varying maximum speed (f) database size



(g) varying τ (∀) (h) query extent(∀)

Figure 13.8: Experiments on synthetic data



(a) varying τ

(b) query extent

Figure 13.9: Experiments on real data ($\forall$)

The PST$\tau\forall$ query shows similar performance results (cf. Figures 13.8(g) and 13.8(h)) for the mentioned parameters. For the experiments, we reduced the temporal query extent to 5, since the number of result objects for a PST$\tau\forall$ query is usually much lower than for a PST$\tau\exists$ query with the same query window.

Influencing Variables of ST Diamonds. The size of the spatio-temporal diamonds is generally affected by two parameters. One is the time interval between successive observations, since a larger interval increases the space that can be reached by the moving object between the two observations. For this experiment, the number of time steps between successive observations in the data set was chosen randomly from the intervals on the x-axis in Figure 13.8(d). The second parameter is the speed of the object and has a similar effect. The speed corresponds to the parameter ϵ, which reflects the maximum distance of points which can be reached by an object within one time step (cf. Figure 13.8(e)). Since larger spatio-temporal diamonds usually result in more objects which intersect the query window, the number of candidates to be refined increase when increasing these two parameters. Interestingly, the effectiveness of the ST-Diamond Filter decreases over the ST-MBR-Filter, whereas the pruning effectiveness of the Probabilistic Filter increases. This shows, that the probabilistic filter copes better with more uncertainty in the data than the other two filters.

Database Size. We evaluated the scalability of the UST-tree by increasing the amount of observations (cf. Figure 13.8(f)). The number of results increases linearly with the database size. The experiment also shows that the number of refined candidates increase linearly.

Real Data. The experiments on the real world data, show similar behavior as those on the synthetic data. Thus we only show excerpts from the evaluation. Figure 13.9(a) illustrates the results for PST$\tau\forall$ queries when varying the value of τ. It is notable that the ST-Diamond Filter seems to even perform better (compared to the ST-MBR-Filter) on the real dataset. The reason for this is that the real dataset has much more inherent irregularity (regarding the locations and the movement of obejcts). This favors the ST-Diamond filter over the MBR approximation (since diamonds are more skewed as in Figure

13.2(b)). The probabilistic filter is apparently not affected. When varying the query extent (cf Figure 13.9(b)) the results resemble the results on the synthetic dataset.

13.6 Conclusions

In this chapter, we proposed the UST-tree which is an index structure for uncertain spatio-temporal data. The UST-tree adopts and incorporates state-of-the art techniques from several fields of research in order to cope with the complexity of the data. We showed how the most common query types (spatio-temporal ∃- and ∀-window queries) can be efficiently processed using probabilistic bounds which are computed during index construction. To the best of our knowledge, this is the first approach that supports query evaluation on very large uncertain spatio-temporal databases, adhering to possible worlds semantics. Outside the scope of this chapter is the consideration of an object's location before its first and after its last observation. In both cases, the resulting diamond approximation would be unbounded. An approach to solve this problem is to define a maximum time horizon for which diamond approximations are computed. Beyond this horizon, we can use the stationary distribution of the model M to infer the location of an object.

Chapter 14

Sampling and Similarity Search on Uncertain Spatio-Temporal Data

The preceding chapters of this part focused on probabilistic spatio-temporal window queries on uncetain spatio-temporal data. In this chapter, we address the problem of similarity search in a database of uncertain spatio-temporal objects. Each object is defined by a set of observations ((time,location)-tuples) and a Markov chain which describes the objects uncertain motion in space and time as discussed in the previous two chapters. To model similarity, we extend the definition of the well-known Longest Common Subsequence (LCSS) measure to the case of uncertain spatio-temporal data (ULCSS). We show how the aligned version (without time shifting) of the ULCSS can be exactly computed in PTIME. In addition, we show how to use a sampling approach to approximate the general ULCSS. For this purpose, an efficient solution for sampling a possible world (trajectory) is necessary. Therefore we develop a method that is based on Bayesian inference to incorporate the observations into the model. In addition to approximating the ULCSS, our approach can also approximate the results of a broad class of queries over uncertain spatio-temporal data. Parts of this chapter have been published in [66].

The rest of this chapter is organized as follows: Section 14.1 gives an introduction to similarity search in uncertain spatio-temporal databases and provides several application scenarios. We will extend the distance measure LCSS to an uncertain setting in Section 14.2. Section 14.3 reviews related work. Section 14.4 presents an exact polynomial-time algorithm to compute a special case of the ULCSS, where the allowed amount of time-shifting is set to zero. To solve the problem of the prohibitively large computational complexity of the exact solution to the general ULCSS, in Section 14.5 we develop a sampling algorithm. In addition, we show how to apply our sampling algorithm to other queries on uncertain trajectories based on the Markov model, demonstrating the wide applicability of our approach (Section 14.7). Finally, we conduct an experimental evaluation on both synthetic and real datasets in Section 14.6 and conclude the chapter in Section 14.8.

14.1 Introduction

Similarity search on trajectory data has an increasing number of applications, especially after the widespread availability of location data, such as GPS tracks. Data analysis tasks such as identifying frequent motion patterns or trajectory clustering require finding objects that moved in a similar way or followed a certain motion pattern. Furthermore, similarity search on trajectory data gives support to a variety of applications such as traffic routing, logistics, emergency handling, drive guiding systems and flow analysis [157].

A number of similarity measures have been proposed for trajectory data. Among these, the Longest Common Subsequence (LCSS) has been reported less sensitive to noise compared to other distance functions such as DTW and Euclidean distance [162]. The LCSS between two trajectories (i.e., moving objects) can be interpreted as the maximum amount of time the two objects were located at the same position. Further relaxations of this basic LCSS version can be made; i.e. two trajectory positions can be assumed to be equal if their spatial distance by at most ε and their time difference is at most δ.

Most of the previous work on similarity search in trajectory databases assumes the data to be certain or deterministic, which is not the case in many real applications. For example, even though we can get snapshots of the positions of a mobile object through RFID technology, the trajectory data is incomplete and therefore uncertain, since the locations of the object between two consecutive RFID readers are unknown. Other tracking and positioning methods such as GPS and odometry introduce similar problems due to the uncertainty of the underlying signal. Clearly, the previous work for certain trajectory data cannot be applied in the presence of uncertainty. Therefore, it is essential to develop new techniques to find similar trajectories on uncertain data.

In this chapter, we model uncertain trajectories by a Markov model as suggested in Chapter 12, where an object, located at a given spatial cell at time t can transition into other cells at the next timestamp with given transition probabilities. Therefore, an uncertain object may not be located at a given spatial location at a given point in time but at several possible locations instead. To handle data uncertainty, we have to take all of these locations into account. Since an uncertain trajectory may include an exponential number of possible worlds, it becomes challenging to compute the similarity between uncertain trajectories. The simple approach of replacing an uncertain trajectory by an exact trajectory, where the in-between positions are computed by interpolation may yield an impossible world (e.g., running through a lake), which is unacceptable. Alternatively, using a single possible trajectory (e.g. the most probable trajectory), may lead to inaccurate results, since even the most probable trajectory could have a very low probability.

In particular we study how the LCSS can be extended to apply on uncertain data (ULCSS). Since in our scenario the exact motion of an object is unknown (i.e. multiple trajectories of an object become possible) we can model ULCSS as a distribution of all possible LCSS (considering all possible worlds in the uncertain data).

Applications LCSS over uncertain data has many applications. For example, it can be used to evaluate the spread of flu or other diseases. Suppose that an object was diagnozed with a serious communicable disease (*source object*). To curtail this disease, the health

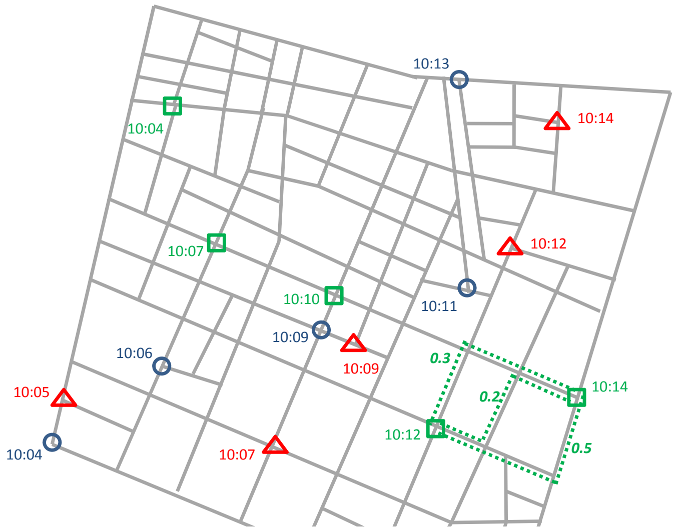


Figure 14.1: Uncertain trajectories.

authorities may want to identify all individuals possibly been infected by the source object. They know that enough virus cells are transmitted between two individuals, if the two persons share the same location for at least k points in time. Additionally, individuals might not even have to meet spatially, since germs can also be transmitted over air. To identify all individuals that might have been infected, the authorities could run an ULCSS query to find all objects having a large enough probability of being at the same location as the source object for at least k points in time.[1] As a example of the above scenario, consider the observations of three objects with their corresponding timestamps, as shown Figure 14.1. Assume that the observations of the object which is the source of the infection is illustrated by the triangles. By taking only these sparse observations into account, we might not be able to attribute accurate probabilities on whether the other two objects (illustrated by circles and rectangles) could have been infected by the source object. Instead we should consider all the possible paths of the uncertain objects with their corresponding probabilities (exemplary shown by the three alternative trajectories between the two last observations of the object illustrated by the green rectangles).

Another application for ULCSS (and trajectory similarity search in general) is suspect identification. Consider a database with (uncertain) trajectories of users in a city at a particular day when a crime took place. An authority (e.g., the police) which has access to the database can identify the trajectories (and identities) of suspects by finding the most similar trajectories to the criminal's route; if a trajectory has a long subsequence which is temporally and spatially close to the criminal's route, this may indicate an exchange between the corresponding person and the criminal, which renders the person a suspect.

Other applications include finding "popular routes", traffic pattern analysis [104], and

---

[1] Google Flu Trends has a similar goal; it employs the amount of flu-related search terms queried at a given location to estimate flu activity. However, in contrast to our approach, Google Flu Trends does not take the motion of objects into account.

trajectory clustering. Traffic data are often given by a sequence of observations of cars such as those illustrated in Figure 14.1. Since the trajectories are uncertain, ULCSS can be used to identify similar movements with a high probability.

14.2 Problem Definition

Again we rely on the model described in Chapter 12. In contrast to the previous Chapter we assume the general case where observations may not only be restricted to one state but may be a *pdf* over the state space. Assume we are given two database objects o_1 and o_2, each of them given by a set of observations Θ^{o_1} and Θ^{o_1} and a transition matrix $M^{o_1}(t)$ and $M^{o_2}(t)$, respectively. Our goal is now to assess the similarity between these two uncertain objects.

Definition 14.1 (LCSS[162]). Let A and B be two trajectories of moving objects with size n and m respectively, where $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_m)$. Let $Head(A) := (a_1, \ldots, a_{n-1})$. Given an integer δ and a real number ϵ, the Longest Common Subsequence is defined as follows: $LCSS_{\delta,\epsilon}(A, B) :=$

$$\begin{cases} 0 \ \text{ if } A = \emptyset \text{ or } B = \emptyset, \text{else} \\ 1 + LCSS_{\delta,\epsilon}(Head(A), Head(B)) \\ \quad \text{if } dist(a_n, b_m) < \epsilon \text{ and } |n - m| \leq \delta \\ max(LCSS_{\delta,\epsilon}(Head(A), B), LCSS_{\delta,\epsilon}(A, Head(B))) \\ \quad otherwise \end{cases}$$

Parameter ϵ constraints the spatial distance between two locations in order to match in space; in many applications, objects rarely visit the same locations, but being "close enough" is equivalent to meeting. Parameter δ controls how far in time we can expand in order to match a given point from one trajectory to a point in another trajectory; in many applications (like the flu spreading problem discussed in the Introduction), two trajectories of A and B may still be considered similar, even if they do not visit positions at the same time. For instance, in an application where it is required to find objects which follow a similar route then the similarity of two trajectories should not decrease if two objects vary their speed on the same route. In [162], LCSS has been reported less sensitive to noise compared to other similarity measures for multi-dimensional time series, like Dynamic Time Warping and Euclidean distance.

In this work, we aim at computing $LCSS_{\delta,\epsilon}(o_1, o_2)$ for two uncertain trajectories o_1 and o_2. According to possible world semantics, the result of $LCSS_{\delta,\epsilon}(o_1, o_2)$ is not longer a single scalar, but rather a probability density function on $\mathbb{N}$, mapping each possible outcome $k \leq min(length(o_1), length(o_2))$ to a probability $P(LCSS_{\delta,\epsilon}(o_1, o_2) = k)$.

Definition 14.2 (ULCSS)**.** Let o_1 and o_2 be two uncertain trajectories. The Uncertain Longest Common Subsequence (ULCSS) between o_1 and o_2 is a random variable, defined by the following probability density function:

$$\mathrm{ULCSS}_{\delta,\epsilon}(o_1, o_2) : \mathcal{D} \times \mathcal{D} \to (\mathbb{N} \to [0,1] \in \mathbb{R})$$

$$\mathrm{ULCSS}_{\delta,\epsilon}(o_1, o_2) := \mathit{pdf}(x \in \mathbb{N}) = P(\mathrm{LCSS}_{\delta,\epsilon}(o_1, o_2) = x)$$

In addition, we study the special case of ULCSS, where $\delta = 0$; for this case, we propose a PTIME algorithm for its exact computation:

Definition 14.3 (UALCSS)**.** Let o_1, o_2 be two uncertain trajectories. The Uncertain Aligned Longest Common Subsequence is defined by $\mathrm{UALCSS}_{\epsilon}(o_1, o_2) = \mathrm{ULCSS}_{0,\epsilon}(o_1, o_2)$.

To quantize the similarity between two uncertain trajectories, we can use the *expected* ULCSS: $\sum_{x} P(\mathrm{ULCSS} = x) * x$. In addition, a threshold based approach can be used (e.g., $P(\mathrm{ULCSS} \geq \phi)$).

14.3 Related Work

Similarity search in trajectory databases has received plenty of attention (e.g. see [162] and its follow-up work). However, for the case where the trajectories of objects are uncertain, there exists, to the best of our knowledge, no work so far which is in accordance to the possible worlds semantics. Therefore, in this section, we review the works related to similarity search using Markov chains as a model and query processing in uncertain spatio-temporal databases.

Similarity Search and Markov Chains. Due to their representation power and relatively simple computation, Markov Chains and their variants have been used in various applications. [36] addressed the problem of estimating the probability of finding a word of length k common between r of q strings generated by a Markov process. This work is different from ours since we want to compute the probability *distribution* over the length of arbitrary common substrings produced by two Markov processes with possibly different transition properties. [11] provided approximations for the length of the (unaligned) longest common subsequence on the Markov model under the assumption that the underlying Markov chains are aperiodic and irreducible. [91] addressed a similar problem as [11]; the authors assumed the transition matrices to be at least similar. Probabilistic analyses such as [11, 91] usually investigate the asymptotic probability of the longest common substring/subsequence, i.e. probabilities where the length of the underlying sequences is large, which is not necessarily the case in our scenario. [75] aimed at matching subsequences onto probabilistic time series described by a hidden semi-Markov-model. The use of the semi-Markov-model, initially applied to speech recognition, enables the possibility of time shifting, i.e. matching different points in time. Another common application area of hidden Markov models is in the area of bioinformatics where gene sequences have to be matched (e.g. [84] and the references therein). When employing hidden Markov models, viterbi-like algorithms [71] and extensions that can handle insertions and deletions (c.p. [8], [84], [100]) are usually employed for computing the maximum likelihood, and not a distribution over the possible outcomes. Besides their advantage of estimating the *edit distance* between sequences, these approaches can only be used to match sequences to Markov chains but not two Markov chains.

Furthermore, similar to our algorithm of adapting transition matrices is the Baum-Welch algorithm for hidden Markov chains [165]. This algorithm aims at estimating *time-invariant* transition matrices and emission probabilities of a hidden Markov model when these probabilities are not known. In contrast, we assume this underlying model to be given, however we aim at adapting it to allow efficient sampling by computing *time-variant* transition matrices. Therefore, the Baum-Welch algorithm, following the well-known EM-paradigm is approximate and iterative in nature, converging to the true distribution while ours is exact.

Related to our algorithm is also the Forward-Backward-Algorithm for Hidden Markov models, but this algorithm aims at computing the state distribution of a Markov chain for each point in time, given an observation. In contrast, we aim at computing transition matrices for each point in time, given a set of observations.

Similarity Search in Spatio-Temporal Databases. The chapter addresses the problem of trajectory similarity, as e.g. addressed in [131] (and the extensive references therein) – given their classification, we investigate the problem of spatio-temporal similarity. Trajectory similarity is often obtained by employing Euclidean Distance, Dynamic Time Warping (DTW), and LCSS. However, all of these approaches mainly adapt to trajectories in a certain setting. [162] relaxed this assumption by taking the effect of noise into account, however different paths of an object cannot be considered equivalent to Markov chains. Uncertain trajectory similarity has been investigated in the context of trajectory classification by [129] employing intuitionistic fuzzy sets. However this work addresses *position* uncertainty instead of *motion* uncertainty. The idea behind this model is that GPS measurements are uncertain such that the position at a given point in time cannot be determined accurately. However, we aim at modeling the behaviour of an object under the condition that neither position nor motion are known at a given point in time, but might have been known at some time in the past. Recently, [109] addressed the problem of computing the windowed LCSS on randomly generated strings. However, the different characters in a word are drawn independently in this context, whereas the state at time t depends on the previous state in our problem setting.

14.4 Uncertain Aligned Longest Common Subsequence

LCSS was originally proposed to model similarity between strings. In this case there does not exist a parameter ϵ, since the characters of the strings have to match exactly. An open problem in applied probability is the following: compute the longest common subsequence shared by two sequences (or strings) of length N, whose elements are chosen

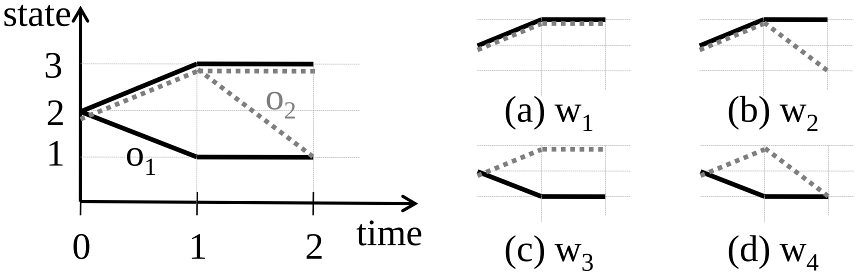


Figure 14.2: The possible paths and possible worlds $\{w_{1-4}\}$ of two uncertain objects. Possible worlds have a probability of 0.25 each.

at random from a finite alphabet. In particular, calculating the exact distribution and the expected value of the length of the LCSS between two random sequences is NP-hard [72]. The above problem however is a special case of the one tackled in this work, since the probability for each character is independent from the previous character, which is not the case in uncertain spatio-temporal data, making our problem of computing ULCSS even more difficult.

Nevertheless, while it remains unsolved how to compute the exact ULCSS in the general case, in this section we show how to exactly compute the UALCSS (ULCSS for the special case of $\delta = 0$) between two uncertain spatio-temporal objects, which represents the pdf over all possible length of the LCSS between the two evaluated objects with a polynomial time algorithm. UALCSS is relevant to many applications, like the infection application mentioned in the Introduction; virus particles in a droplet infection can only be spread through space, but not through time.

Figure 14.2 (left) shows the uncertain trajectory of objects o_1 (represented by the solid line) and o_2 (represented by the dotted line). From these, we derive four possible worlds as illustrated in Figure 14.2 (right). We can see that in w_1 the (certain) LCSS equals 3, in worlds w_2 and w_4 LCSS=2, while in w_3 LCSS=1. If we assume, in this example, that for each object each alternative trajectory has a probability of 0.5, we get a probability vector [0,0.25,0.5,0.25] for UALCSS, where the kth element in the list denotes k hits between two paths. Clearly, such an approach of enumerating all possible worlds and aggregating their probabilities is not a viable option, since in general, the number of possible trajectories of an uncertain trajectory is exponential in the length of the uncertain trajectory. Thus, the number of possible worlds is also exponentially large.

14.4.1 Computation of UALCSS

For computing UALCSS, we have to take certain dependencies of the two objects into account, i.e., the relative position of o_1 to o_2 at time $t = 0$ (w.l.o.g. we assume that the first observations of o_1 and o_2 are at time $t = t_1^{o_1} = t_1^{o_2} = 0$) will affect the length of the UALCSS at a later time $t > 0$. For this reason, we have to take the conditional probabilities of object o_1 being in state s_i when o_2 is in state s_j into account: At the initial time $t = 0$ we assume both objects locations to be independent, and therefore we can write $P(o_1(0) = s_i \wedge o_2(0) = s_j) = P(o_1(0) = s_i) \cdot P(o_2(0) = s_j)$. Let $T(t)$ be a probability matrix with $T_{ij}(t) = P(o_1(t) = s_i \wedge o_2(t) = s_j)$, denoting that the corresponding objects are within state s_i and s_j at time t. The matrix $T^0(0)$ can be easily computed as follows (the superscript 0 denotes the numbers of hits gained so far): $T^0(0) = P(o_2, 0) \cdot P(o_1, 0)^T$. This is the case because we have $T^0_{ij}(0) = P_i(o_1, 0) \cdot P_j(o_2, 0) = P(o_1(0) = s_i) \cdot P(o_2(0) = s_j)$.

The elements $T^0_{ii}(0)$ denote the probabilities that both uncertain objects are located within the same state i at time 0, increasing the longest common subsequence by 1, such that these *possible worlds* have to be marked. This can be simply achieved by moving them into a second matrix $T^1(0)$, where $T^1_{ii}(0) = T^0_{ii}(0)$ and $T^1_{ij}(0) = 0$ for $i \neq j$. Besides, the shifted elements have to be deleted from $T^0(0)$ by performing $T^0_{ii}(0) = 0$. Now both matrices contain possible worlds, split by their number of hits.

After initialization, this method can be applied in a similar manner to compute the equivalence classes of possible worlds within each time $t \neq 0$, which is achieved by updating all state matrices $T^k(t-1)$. As a first step, the states of o_1 and o_2 in $T^k(t-1)$ have to be transitioned. Given a state vector $P(o_i, t-1)$, this transition is usually performed by multiplying $P(o_i, t-1)$ with its corresponding, pre-determined, transition matrix $M^{o_i}(t-1)$, i.e., $P(o_i, t) = M^{o_i}(t-1)^T \cdot P(o_i, t-1)$. However, in our scenario, we do not have a single state vector, but a state matrix $T^k(t-1)$, containing conditional probabilities of both objects. The elements in this matrix have to be transitioned according to both transition matrices $M^{o_1}(t-1)$ and $M^{o_2}(t-1)$. It can be proven that:

$$T^k(t) = M^{o_1}(t-1)^T \cdot T^k(t-1) \cdot M^{o_2}(t-1)$$

Proof. Let $M^k(t-1)$ at time $t-1$ be given. The goal is to compute $M^k(t)$, i.e., M^k with transitions applied to o_1 and o_2 for time t.

$$T^k_{ij}(t) = P(o_1(t) = s_i \wedge o_2(t) = s_j | t)$$

$$= \sum_{s_x \in \mathcal{S}} \sum_{s_y \in \mathcal{S}} P(o_1(t-1) = s_x \wedge o_2(t-1) = s_y) \cdot M^{o_1}_{xi}(t-1) \cdot M^{o_2}_{yj}(t-1)$$

$$= \sum_{s_x \in \mathcal{S}} M^{o_1}_{xi}(t-1) \cdot \sum_{s_y \in \mathcal{S}} P(o_1(t-1) = s_x \wedge o_2(t-1) = s_y) \cdot M^{o_2}_{yj}(t-1)$$

$$= \sum_{s_x \in \mathcal{S}} M^{o_1}_{xi}(t-1) \cdot \sum_{s_y \in \mathcal{S}} T^k_{xy}(t-1) \cdot M^{o_2}_{yj}(t-1)$$

$$\Leftrightarrow M^{o_1}(t-1)^T \cdot M^k(t-1) \cdot M^{o_2}(t-1)$$

□

After performing the transition, the matrix element $T^k_{ij}(t)$ again contains the conditional probabilities at time t that o_1 is in state j while o_2 is in state i. After transitioning, the hits are extracted from the matrix, by shifting the diagonal elements to $T^{k+1}_{ii}(t-1)$ and removing them from $T^k_{ii}(t-1)$. Therefore each of the t transitions leads to at most one additional matrix; thus, the total space complexity of this algorithm is at most $O(|S|^2 \cdot t)$ and the runtime complexity is $O(\Delta t^2 \cdot |\mathcal{S}|^3)$. In practice, these costs are much lower since vectors $P(o_i, t)$ and $M^{o_i}(t)$ are both sparse and we can save space and computations by employing sparse matrix operations on compressed representations.

After having completed t transitions, we can derive the probability distribution for the relative frequency of worlds that had a given number k of hits:

$$P(|\{x \in \mathcal{T} | o_1(x) = o_2(x)\}| = k) = \sum_{\forall i,j} T_{ij}^k(t)$$

The formal description of this algorithm in pseudocode can be found in Algorithm 12. Incorporating observations (function *reweight()* in Algorithm 12) is generally similar to incorporating observations in the sampling-based approach described later on. Let us first assume that o_1 was observed at state i. Then all columns $j \neq i$ in M^k have to be set to 0 and all matrices have to be reweighted such that $\sum_{x \in t} M^k = 1$. Accordingly, if o_2 has been observed at a given state, the corresponding rows have to be set to zero. If observations are uncertain, the rows and columns corresponding to an observation must be multiplied by the probability of the observation happending in exactly this row/column, and be reweighted accordingly. Furthermore, the algorithm can be easily adapted for $\epsilon > 0$. In this case, not only diagonal elements from M^k have to be shifted, but also further matrix elements that correspond to locations with a distance $\leq \epsilon$ to a given location of an uncertain object.

Algorithm 12 UALCSS(o_1, o_2, t_{max})

```
$T^0 = P(o_2, 0) \cdot P(o_1, 0)^T$
$T^1_{i \neq j} = 0$
$T^1_{ii} = T^0_{ii}$
$T^0_{ii} = 0$
for $t = 1; t \leq t_{max}; t++$ do
    for $k = t; k \geq 0; k--$ do
        $T^k = M^{o_1}(t-1)^T \cdot T^k \cdot M^{o_2}(t-1)$
        $T^{k+1}_{ii} = T^{k+1}_{ii} + T^k_{ii}$
        $T^k_{ii} = 0$
    end for
    if $\exists t_i^{o_1} : t_i^{o_1} = t \vee \exists t_j^{o_2} : t_j^{o_2} = t$ then
        reweight($\{M^k\}$, $\theta_i^{o_1}$, $\theta_j^{o_2}$)
    end if
end for
$p = Array[t_{max} + 1]$
for $t = 0; t \leq t_{max}; t++$ do
    $p_k = |T^k|_{L_1}$
end for
return $p$
```

14.4.2 Example

Let us exemplarily compute the UALCSS for the Markov chains from Figure 14.2 for the time interval $t = [0, 2]$. The initial state vector is $o_1(0) = o_2(0) = (0, 1, 0)^T$ for both uncertain objects. From this we derive the conditional probability of an object being in one state if the other object is in the other state by evaluating Line 1, yielding

$$T^0(0) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

The resulting matrix denotes that both objects are within state 1 at $t = 0$ for sure, leading to a hit such that $T^1(0) = T^0(0)$ and $T^0(0) = 0$. The resulting matrices $T^0(0)$ and $T^1(0)$ denote the set of possible worlds with zero and one hits at time $t = 0$, respectively.

From Figure 14.2 we can derive the transition matrices for the two uncertain objects; these will be used during the next point in time. Again, the entries of a transition matrix, M_{ij}, denote the probability that an uncertain object changed its state from state i to state j at a given round. Given that the probability of entering a branch in 14.2 is uniformly distributed over all branches, the transition matrix can be directly derived from the figure:

$$M^{o_1} = \begin{pmatrix} 1 & 0 & 0 \\ 0.5 & 0 & 0.5 \\ 0 & 0 & 1 \end{pmatrix}, M^{o_2} = \begin{pmatrix} 0.5 & 0 & 0.5 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

For the sake of brevity, we will not evaluate both loops from the pseudocode separately. At time $t = 1$, we first have to transition the set of possible worlds, yielding

$$T^0(1) = T^0(0), T^1(1) = \begin{pmatrix} 0.5 & 0 & 0.5 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \text{(Line 2-4)}$$

The semantic of this matrix is that object o_2 is in state 3 for sure. Furthermore o_1 is in state 1 or 3 with a probability of 0.5. By evaluating lines 8-9, we can then derive the probability distribution over the number of hits at time $t = 1$:

$$T^0 = 0, T^1 = \begin{pmatrix} 0 & 0 & 0.5 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, T^2 = \begin{pmatrix} 0.5 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

At the next point in time, we again transition the three matrices based on the two transition matrices, resulting in:

$$T^0 = 0, T^1 = \begin{pmatrix} 0 & 0 & 0.25 \\ 0 & 0 & 0 \\ 0 & 0 & 0.25 \end{pmatrix}, T^2 = \begin{pmatrix} 0.25 & 0 & 0 \\ 0 & 0 & 0 \\ 0.25 & 0 & 0 \end{pmatrix}$$

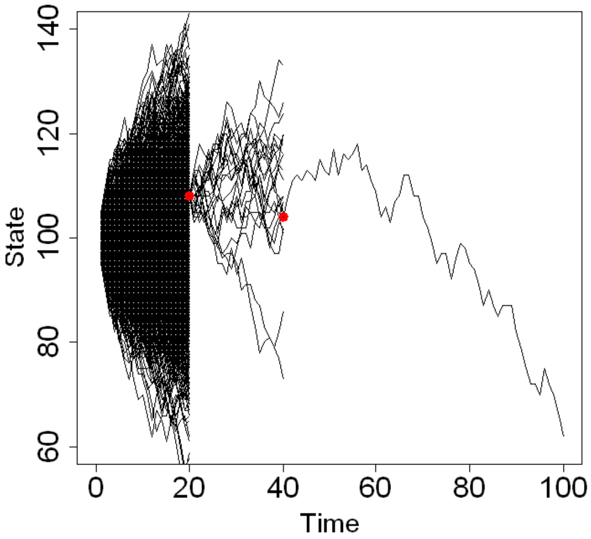


Figure 14.3: Traditional MC-sampling.

We again shift the diagonal matrix of each object by one to the right, resulting in:

$$T^0 = 0, T^1 = \begin{pmatrix} 0 & 0 & 0.25 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, T^2 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0.25 & 0 & 0.25 \end{pmatrix}, T^3 = \begin{pmatrix} 0.25 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Finally, we can subsume the probability in each matrix to determine the distribution of the aligned longest common subsequence, resulting in $p = [0, 0.25, 0.5, 0.25]$

14.5 Sampling Possible Trajectories

As discussed in 14.4 computing ULCSS in the general case is NP-hard. However, in most applications it is sufficient to compute an approximate ULCSS. For this purpose, in this section, we propose a sampling approach, tailored to uncertain spatio-temporal data. We first show that a traditional (naïve) sampling approach is not applicable for our problem, as it does not account for all observations of an object, creating a very large number of sample paths, which are not possible given all observations. To tackle this issue, we employ a different approach, which incorporates information about observations directly into the Markov model, following a forward-backward paradigm. Our approach uses Bayesian inference in order to iteratively adapt the parameters of the model, given each observation. Based on the resulting adapted model, again a (time inhomogeneous) Markov chain, we can perform traditional sampling in order to ensure samples that are consistent with all observations, and therefore guarantee that the probability of drawing each sample is equal to the true probability of the corresponding trajectory.

14.5.1 Traditional Sampling

To sample possible trajectories of an object, a traditional Monte-Carlo approach starts by taking the first observation of an object, then transitioning to new states according to the distribution given by the underlying transition matrix. This approach however, cannot directly account for additional observations, as illustrated in Figure 14.3. Here, we assume (for simplicity) a one dimensional space, and an object randomly transitioning to adjacent states at each point of time. In the plot, 1000 samples are initiated at the first observation at time $t = 0$ and transition according to the model. Given the second observation at time $t = 20$, a number of trajectories become inconsistent (i.e., impossible given this second observation), because these trajectories do not conform to both the observation and the motion constraints of the object. Such impossible trajectories are no longer expanded to further states in Figure 14.3. At time $t = 40$, even more trajectories become invalid; in the end, only one out of a thousand samples remains possible and useful.

Clearly, the number of sample trajectories required to obtain a single valid trajectory increases exponentially in the number of observations of an object, making this traditional Monte-Carlo approach inappropriate. In the next section, we will show how to obtain possible samples efficiently, for an arbitrary number of observations. The main challenge here is not only to ensure that only possible trajectories are sampled, but also that the probability of a sampled path corresponds to the true probability of the object taking this path, given the initial model and all observations.

14.5.2 Adapting the Model to Observations

The traditional sampling approach essentially uses the transition probabilities $P(o(t+1) = s_j|o(t) = s_i)$ given by the Markov chain to create sample trajectories. To incorporate the knowledge given by a set of observations Θ^o of an object o, we need to consider the probability

$$P(o(t+1) = s_j|o(t) = s_i, \Theta^o),$$

that is the probability that object o transitions to state s_j from state s_i at time t, given all observations. The Markov property yields:

$$P(o(t+1) = s_j|o(t) = s_i, \{\theta^o_m|t_m \geq t\})$$

This is derived by assuming that o is located at s_i at time t, and then computing the probability of visiting state s_j at time $t+1$. This can be computed using an $\exists$-window query as defined in Definition 12.2. An $\exists$-window query returns the probability of intersecting a set of $(time, location)$ pairs (in this case this set contains only the pair $(t+1, s_j)$), given observations in the future. Performing the above computation for each time t between the first and the last observation of o, and for each state pair s_i,s_j yields a new Markov chain, which is adapted to Θ^o. The resulting inhomogeneous Markov chain can be used to draw sample trajectories, which are guaranteed to comply with all observations, and whose probability of drawing this sample corresponds to the true probability of this trajectory

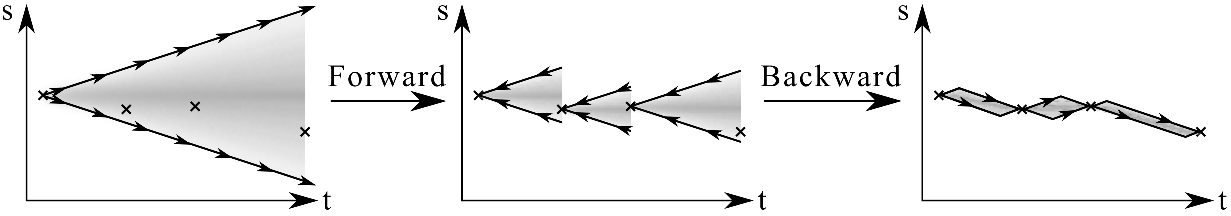


Figure 14.4: An overview over our forward-backward-algorithm.

given information about the initial Markov chain and all observations. [2] Although correct and computable in polynomial time (unlike the naïve sampling approach), this approach still suffers from high computational cost for the construction of the new transition matrices: for each point in time, each entry of the original Markov chains has to be considered and a window query, which runs in $O(|S|^2 \cdot \Delta t)$ (cf Section 12.3.3), has to be performed. Although this has to be done only once, independent of the number of samples, the total time complexity of $O(|S|^4 \cdot \Delta t^2)$ makes this approach inapplicable for real applications, even when sparse matrix and vector operations are exploited.

We now present a different algorithm, which can compute the probabilities $P(o(t+1) = s_j|o(t) = s_i, \Theta^o)$ more efficiently.

14.5.3 Efficient Model Adaption

In a nutshell, this problem can be solved in the well-known forward-backward manner: Starting at the time of the first observation t_1^o with the initial observation θ_1^o, we perform transitions of object o using the original Markov chain of o until the final observation at time $t_{|\Theta^o|}$ is reached. During this *Forward*-run, Bayesian inference is used to construct a time-reversed Markov-model R_t^o of o at time t given observations in the past, i.e., a model that describes the probability

$$R_{ij}^o(t) := P(o(t-1) = s_j|o(t) = s_i, \{\theta_m^o|t_m^o < t)\})$$

of coming from a state s_j at time $t-1$, given being at state s_i at time t and given the observations in the past.

Then, in a second step, the *Backward*-run, we traverse time backwards, from time $t_{|\Theta^o|}$ to t_1, by employing the time-reversed Markov-model $R^o(t)$ constructed in the forward step. Again, Bayesian inference is used to construct a new Markov model $F^o(t-1)$ that is further adapted to incorporate knowledge about observations in the future. This new Markov model contains the transition probabilities

$$F_{ij}^o(t-1) := P(o(t) = s_j|o(t-1) = s_i, \Theta^o). \tag{14.1}$$

for each point of time t, given all observations, i.e., in the past, the present and the future.

---

[2] This claim can be proven simply by applying possible worlds semantics, as shown in Section 12.4

As an example, Figure 14.4 visualizes a one-dimensional uniform random walk, i.e. a very simple Markov chain where the object may move to adjacent states with a uniform distribution. Figure 14.4(a) shows the initial model, using knowledge about the first observation only. In this case, a large set of (*time*,*location*) pairs can be reached with a probability greater than zero. The shading of reachable (*time*,*location*) pairs indicates the likelihood of these pairs, assuming that a detour is less likely than a direct path.[3] The adapted model after the forward phase is depicted in Figure 14.4(b), significantly reducing the space of reachable (*time*,*location*) pairs and adapting respective probabilities, thus drastically improving the model. Since the model of Figure 14.4 (b) is obtained trivially, the main contribution of this section is the backward-phase which adapts the Markov model to observations in the future. This task is not trivial, since the Markov-property does not hold for the future, i.e., the past is *not* conditionally independent of the future given the present. During the backward phase, we traverse the Markov chain backwards, from time $t_{|\Theta^o|}$ to t_1, by employing the information acquired in the forward-phase. This phase yields the final transition matrices for each point in time, such that all observations Θ^o are taken into account for the adapted model. The resulting final transition matrices $F^o(t-1)$ contain the transition probabilities $F^o(t-1)_{ij} = P(o(t) = s_j | o(t-1) = s_i, \Theta^o)$. Figure 14.4(c) shows the resulting final model after the backward phase. Before describing the algorithm in more detail, we prove the following corollary of the Bayes Theorem.

Lemma 14.1 (Conditional Bayes)**.**

$$P(A|B,C) = \frac{P(B|A,C)P(A|C)}{P(B|C)}$$

Proof. By applying the Multiplication Theorem of Probability and due to the commutativity of the conjunction of random events we get:

$$P(A \wedge B \wedge C) = P(C)P(B|C)P(A|B,C)$$

$$\Leftrightarrow \frac{P(A \wedge B \wedge C)}{P(B|C)P(C)} = P(A|B,C)$$

$$\Leftrightarrow \frac{P(A \wedge B \wedge C)}{P(C)} = P(A|C)P(B|A,C)$$

Mutual substitution leads to Lemma 14.1. □

Forward Phase

The main challenge of the forward-phase is to construct necessary data structures for efficient implementation of the backward-phase, namely the time-reversed transition matrix $R^o(t)$ that summarizes the transition probabilities for a transition from time t to time $t-1$.

---

[3] This assumption is only made for illustration of this example. In general, a detour via a highway may be more likely than a direct path through a lake. These likelihoods are captured by the given Markov-model.

This matrix is used to incorporate information about future observations in the backward phase. To obtain $R^o(t)$, we can apply the theorem of Bayes as follows:

$$R^o(t)_{ij} := P(o(t-1) = s_j | o(t) = s_i) = \tag{14.2}$$
$$\frac{P(o(t) = s_i | o(t-1) = s_j) \cdot P(o(t-1) = s_j)}{P(o(t) = s_i)}$$

By assuming existence of all observation $past^o(t) := \{\theta^o_m | t^o_m < t)\}$ of o that occurred before time t, all events become further conditioned to these observations as follows, using Lemma 14.1:

$$R^o(t)_{ij} := P(o(t-1) = s_j | o(t) = s_i, past^o(t)) = \tag{14.3}$$
$$\frac{P(o(t) = s_i | o(t-1) = s_j, past^o(t)) \cdot P(o(t-1) = s_j | past^o(t))}{P(o(t) = s_i | past^o(t))}$$

The probability $P(o(t) = s_i | o(t-1) = s_j, past^o(t))$ can be rewritten as $P(o(t) = s_i | o(t-1) = s_j)$, exploiting the Markov property (cf Equality 12.5).[4] This probability is given by the original Markov-chain $T^o(t)$.

Furthermore, both priors $P(o(t-1) = s_j | past^o(t))$ and $P(o(t) = s_i | past^o(t))$ can be computed in a single run: We start at $t = t_1$ using the initial distribution of θ^o_1. Then, transitions are performed iteratively using the original Markov chain $M^o(t)$. For each intermediate point of time t, all probabilities $P(o(t-1) = s_j | past^o(t))$ are memorized. In each iteration of the forward algorithm, where a new observation $present^o(t) := \theta^o_x, < t^o_x, \theta^o_x > \in \Theta^o, t^o_x = t$ is reached, we incorporate this information to the model. Therefore, we compute $P(o(t) = s_i | past^o(t), present^o(t))$ from $P(o(t-1) = s_j | past^o(t))$ for all states s_i, employing all s_j [5]. This is done by exploiting the assumption that observations are mutually *independent*, i.e. that the error made between two measurements is independent. This allows to compute the probability of the event $Y_j := [o(t) = s_j | past^o(t)] =$ "o is in state s_j at time t given observations before t", *and* the event $Z_i := [o(t) = s_i | present^o(t)] =$"$o$ is in state s_i given the observation θ^o_x at time t" exploiting

$$P(Y_j \wedge Z_i) = P(Y_j) \cdot P(Z_i). \tag{14.4}$$

Clearly, the above joint distribution between states observed due to observations before t and states observed due to the observation at time t allows contradicting observations where both random variables result in different states. Let $Y := [o(t) | past^o(t)]$ and $Z := [o(t) | present^o(t)]$. Then such contradicting worlds can be removed by conditioning the probability of Equation 14.4 events to the event $Y = Z$.

$$P\left(Y_j \wedge Z_i | Y = Z\right) = \frac{P(Y = Z | Y_j \wedge Z_i) \cdot P(Y_j \wedge Z_i)}{P(Y = Z)} \tag{14.5}$$

---

[4] The transition probabilities of object o between time $t-1$ and t are independent of the distribution of o at time $t-1$.

[5] The probability $P(o(t-1) = s_j | past^o(t))$ is given by induction hypothesis, with $P(o(t^o_1) = s_j | \emptyset)$ being the induction start given directly by the first observation θ^o_1 at time $t = t^o_1$. We show how to compute $P(o(t) = s_j | past^o(t))$ from $P(o(t-1) = s_j | past^o(t-1))$, thus completing induction for all times $t^o_1 \leq t \leq t^o_{|\Theta|}$.

In the above equation, the term $P(Y = Z|Y_j \wedge Z_i)$ acts as an indicator function that is one if for the two random variables Y and Z it holds that $Y = Z$, i.e., if the observations are non-contradicting, and zero otherwise. The term $P(Y_j \wedge Z_i)$ can be rewritten to the product of the probabilities of both observations to materialize to s_j according to Equation 14.4. Finally, the denominator $P(Y = Z)$ corresponds to the total probability that both observations are non-contradicting, and can rewritten as $\sum_i Y_i \cdot Z_i$ also exploiting independence of observations in Equation 14.4.

In summary, the total probability $P(Y_j \wedge Z_i|Y = Z) = P([(o(t) = s_j|past^0(t)] \wedge [o(t) = s_i|present^o(t))]|s_i = s_j)$ that object o is at state s_j at time t can be computed by performing a simple element-wise multiplication between the j'th element of vector $P(o(t) = s_j|past^o(t))(j)$ and the j'th element of vector θ^o_t.

Algorithm 13 AdaptTransitionMatrices(o)

```
 1: {Forward-Phase}
 2: $P(o, t_1) = \theta^o_1$
 3: for $t = t^o_1 + 1$; $t \leq t^o_{|\Theta^o|}$; $t$++ do
 4:     $X'(t) = M^o(t-1)^T \cdot diag(P(o, t-1))$
 5:     $P_i(o, t) = \sum_{j=1}^{|S|} X'_{ij}(t)$
 6:     $R^o(t)_{ij} = \frac{X'_{ij}(t)}{P_i(o,t)}$
 7:     {Incorporate observation}
 8:     $P_i(o, t) = normalize(P_i(o, t) \cdot present(t))$
 9: end for
10: {Backward-Phase}
11: for $t = t^o_{|\Theta^o|} - 1; t \geq t^o_1; t$-- do
12:     $X'(t) = R^o(t+1)^T \cdot diag(P_i(o, t+1))$
13:     $P_i(o, t) = \sum_{j=1}^{|S|} X'_{ij}(t)$
14:     $F^o(t)_{ij} = \frac{X'_{ij}(t)}{P_i(o,t)}$
15: end for
16: {Return modified object; state vectors and changed transition matrices are relevant}
17: return $o$
```

Now that all observations at time t and before are considered, both data structures, the new state vector $P(o, t)$ and the adapted transition matrix $R^o(t)$ can be efficiently derived from the following temporary matrix, computed in Line 4 of Algorithm 13:

$$X'(t) = M^o(t-1)^T \cdot diag(P(o, t-1))$$

The equation is equivalent to a simple transition at time t, except that the state vector is converted to a diagonal matrix first. This trick allows to obtain a matrix describing the distribution of the position of o at time $t - 1$ and t, instead of a simple distribution of the location of o at time t. Formally, each entry $X'(t)_{i,j}$ corresponds to the probability $P(o(t-1) = s_j \wedge o(t) = s_i | past^o(t))$ which is equivalent to the *numerator* of Eq. 14.3.[6] To obtain the denominator of Eq. 14.3 we first compute the row-wise sum of $X'(t)$ in Line 5:

$$\forall i \in \{1..|S|\} : P_i(o,t) = \sum_{j=1}^{|S|} X'(t)_{ij}$$

The resulting vector directly corresponds to $P(o,t)$, since for any matrix A and vector x it holds that $A \cdot x = rowsum(A \cdot diag(x))$. By employing this rowsum operation, only one matrix multiplication is required for computing $R^o(t)$ and $\vec{s^o}(t)$.

Next, the elements of the temporary matrix $X'(t)$ and the elements of $P(o,t)$ can now be normalized in Equation 14.3, as shown in Line 6 of the algorithm:

$$\forall i,j \in \{1..|S|\} : R^o(t)_{ij} = \frac{X'(t)_{ij}}{P_i(o,t)}$$

Here, the resulting reverse probabilities of R are stored directly in the matrix $M^o(t)$, so matrix M^o can be discarded. This optimization allows to reuse the allocated space of matrix $M^o(t)$, since the old transition probabilities are no longer used in the remainder of the algorithm. Finally, possible observations at time t are integrated in Line 8, using Eq. 14.5.

Backward Phase

During the backward phase, we traverse time backwards, to propagate information about future observation back to past points of time, as depicted in Figure 14.4(c), thus terminating our algorithm. This phase is equivalent to the Forward-Phase, except that we do not use the initial matrices but rather the reverse transition matrices $R^o(t)$ created during the forward phase, which contain, for each point of time t, transition probabilities already conditioned to observations at time t and before time t. The main benefit of $R^o(t)$ is to allow transitions backwards in time, since $R^o(t)$ is defined as a transition matrix containing the probabilities of going from one state at time $t+1$ to another state at time t. In a nutshell, $R^o(t)$ is a Markov chain with the time axis being reversed. The following reverse Markov property holds for each element R^o_{ij} of matrix R^o:

$$P(o(t) = s_j | o(t+1) = s_i, o(t+2) = s_{t+2}, ..., o(t+k) = s_{t+k}) =$$
$$P(o(t) = s_j | o(t+1) = s_i) \tag{14.6}$$

As an initial state for the backward phase, we use the state vector $P(o, t^o_{|\Theta^o|})$ that results from Section 14.5.3, by element-wise multiplication of the vector derived by using the Markov-chain and all observation but the final observation and the vector $\theta^o_{|\Theta^o|}$ of the

---

[6] The proof for this transformation $P(A \cap B|C) = P(A|C) \cdot P(B|A,C)$ can be derived analogously to Lemma 14.1.

final observation. This way, we take the final observation as given, making any further probabilities that are being computed conditioned to this observation. At each point of time $t \in [t_{|\Theta^o|}, t_1]$ and each state $s_i \in S$, we compute the probability that o is located at state s_i at time t *given* (conditioned to the event) that the observations $future^o(t) := \{\theta^o_m | t^o_m > t)\}$ at times later than t are made. Using the probabilities of $R^o(t)$ which are already conditioned to the observations $past^o(t)$ and $present^o(t)$, this yields the final transition probabilities $F^o_{ij}(t) := P(o(t) = s_j | o(t+1) = s_i, \Theta^o)$, thus including all observations $\Theta^o = past^o(t) \cup present^o(t) \cup future^o(t)$. To compute these final probabilities, which we need for our sampling approach (c.f. Section 14.5.2), we once again exploit Lemma 14.1:

$$P(o(t+1) = s_j | o(t) = s_i, \Theta^o) =$$

$$\frac{P(o(t) = s_i | o(t+1) = s_j, \Theta^o) \cdot P(o(t+1) = s_j | \Theta^o)}{P(o(t) = s_i | \Theta^o)}$$

By exploiting the reverse Markov property (c.f. Equation 14.6), this becomes equal to:

$$P(o(t+1) = s_j | o(t) = s_i, \Theta^o) = \frac{P_1 \cdot P(o(t+1) = s_j | \Theta^o)}{P(o(t) = s_i | \Theta^o)}, \tag{14.7}$$

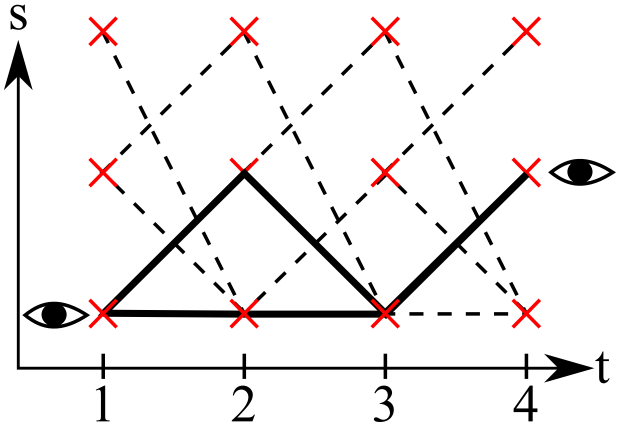
$$\text{where } P_1 = P(o(t) = s_i | o(t+1) = s_j, present^o(t), past^o(t))$$

The probability P_1 is now given by the entries of matrix $R^o(t)$, by its definition. Again, both priors $P(o(t+1) = s_j | \Theta^o)$ and $P(o(t) = s_i | \Theta^o)$ can be computed in a single run: We start at $t = t_{|\Theta^o|}$ using the distribution of $P(o, t^o_{|\Theta^o|})$. Then, transitions are performed backwards until time $t = t_1$ is reached, and for each intermediate point of time t, all probabilities $P(o(t) = s_i, \Theta^o_1)$ are memorized.

Note that even if the initial transition matrix was homogeneous (time-invariant), the resulting transition matrices would typically be inhomogeneous. Also note, that further observations will lead to a higher sparsity of the transition matrices, since new observations may render a large number of (time,state) pairs unreachable, as illustrated in Figure 14.4 (c). Finally, we further note that the transition matrices become more sparse at points of time close to observations. The formal backward algorithm can be found in Lines 11-14 of Algorithm 13 and follows the structure of the forward phase. The returned state vectors $P(o, t)$ can be employed as an initial distribution for starting sampling at any arbitrary point in time.

The overall complexity of this algorithm is $O(\Delta t |\mathcal{S}|^2)$. The initial matrix multiplication requires $|\mathcal{S}|^2$ multiplications. While the complexity of a matrix multiplication is usually in $O(|\mathcal{S}|^3)$, the multiplication of a matrix with a diagonal matrix, i.e., $M^T \cdot s$ can be rewritten as $M^T_i \cdot s_{ii}$, which is actually a multiplication of a vector with a scalar, resulting in an overall complexity of $O(\mathcal{S}^2)$. Rediagonalization needs $|\mathcal{S}|^2$ additions as well, such as renormalizing the transition matrix, yielding $3 \cdot \Delta t \cdot |\mathcal{S}|^2$ for the forward phase. The backward phase has the same complexity as the forward phase, leading to an overall complexity of $O(\Delta t |\mathcal{S}|^2)$.

Example 14.5.1. To make the algorithm more clear, consider the following example from Figure 14.5(a), consisting of four consecutive timestaps and three states. The initially



(a)

$$T = \begin{pmatrix} 0 & 0 & 1 \\ \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & \frac{4}{5} & \frac{1}{5} \end{pmatrix}$$

(b)

Figure 14.5: Exemplary Markov Chain, Visualization (a) and Transition Matrix (b).

time-homogeneous transition matrix, visualized as dashed lines in Figure 14.5(a), is given in Figure 14.5(b). Furthermore, let two observations $\theta_1^o = (0,0,1)^T$ and $\theta_2^o = (0,1,0)^T$ be given. Clearly, given these observations, there is a total of only two possible trajectories (the black lines): $p_1 = (s_3, s_3, s_3, s_2)$ and $p_2 = (s_3, s_2, s_3, s_2)$. The probabilities of these pathes is $P(s_3, s_3, s_3, s_2) = 0.2 \cdot 0.2 \cdot 0.8 = 0.032$ and $P(s_3, s_2, s_3, s_2) = 0.8 \cdot 0.5 \cdot 0.8 = 0.32$. Thus, given the new observations, the probability of trajectory p_1 equals $\frac{0.032}{0.032+0.32} = \frac{1}{11}$ and the probability of trajectory is $\frac{0.32}{0.032+0.32} = \frac{10}{11}$. Albeit correct, it is impractical to compute this exact result in practise, as it requires to enumerate the exponentially large set of all possible trajectories. Furthermore, as discussed in this Section, a naive sampling approach is not viable, as in practice, only an exponentially small number of samples will hit all observations. Thus, we show in the following how to adapt the Markov model of this example, to allow drawing samples that are guaranteed to draw samples conform to the observations, without incurring any bias of the distributions of these correct samples.

To generate the adapted transition matrices that can only produce the two valid trajectories denoted within the picture, we run the algorithm introduced above.

Forward. First, we multiply the transposed transition matrix M with the initial diagonalized observation:

$$X'(2) = \begin{pmatrix} 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{4}{5} \\ 1 & \frac{1}{2} & \frac{1}{5} \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & \frac{4}{5} \\ 0 & 0 & \frac{1}{5} \end{pmatrix}$$

$$\Rightarrow P(o,2) = \begin{pmatrix} 0 \\ \frac{4}{5} \\ \frac{1}{5} \end{pmatrix}, R(2) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

The vector $P(o,2)$ denotes the distribution of the object after the first transition, $R(2)$ the backward transition probabilities of the object. In the following iteration, we build upon $P(o,1)$ and get:

$$X'(3) = \begin{pmatrix} 0 & \frac{2}{5} & 0 \\ 0 & 0 & \frac{4}{25} \\ 0 & \frac{2}{5} & \frac{1}{25} \end{pmatrix} \Rightarrow P(o,3) = \begin{pmatrix} \frac{2}{5} \\ \frac{4}{25} \\ \frac{11}{25} \end{pmatrix}, R(3) = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & \frac{10}{11} & \frac{1}{11} \end{pmatrix}$$

The next iteration works equivalently:

$$X'(4) : \begin{pmatrix} 0 & \frac{2}{25} & 0 \\ 0 & 0 & \frac{44}{125} \\ \frac{2}{5} & \frac{2}{25} & \frac{11}{125} \end{pmatrix} \Rightarrow R(4) : \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ \frac{50}{71} & \frac{10}{71} & \frac{11}{71} \end{pmatrix}, P(o,4) = \begin{pmatrix} \frac{2}{25} \\ \frac{44}{125} \\ \frac{71}{125} \end{pmatrix}$$

Since at $t = 3$ we have an observation, we incorporate it by piecwise multiplication and normalization, getting $P(o,3) = P(o,3)) \cdot \theta_2^o = (0,1,0)^T$

Backward. The backward phase is quite similar to the forward phase, however we reuse the reverse transition matrices computed during the forward phase. We get:

$$X'(3) = R(4)^T \cdot diag(P(o,4)) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

$$\Rightarrow F(3) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}, P(o,3) = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

$$X'(2) : \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & \frac{10}{11} \\ 0 & 0 & \frac{1}{11} \end{pmatrix} \Rightarrow F(2) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}, P(o,2) = \begin{pmatrix} 0 \\ \frac{10}{11} \\ \frac{1}{11} \end{pmatrix}$$

$$X'(1) : \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & \frac{10}{11} & \frac{1}{11} \end{pmatrix} \Rightarrow F(1) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & \frac{10}{11} & \frac{1}{11} \end{pmatrix}, P(o,1) = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

The matrices $F^o(i)$ computed during the backward phase are returned as the set of adapted, now time-inhomogeneous transition matrices.

Sampling Process. Once the transition matrices for each point of time have been adapted to incorporate knowledge about all observations, the actual sampling process is simple: For each object o, each sampling iteration starts by sampling an initial position $P(o,t_1)$ as a random realization of the random variable defined by the initial distribution of the object's first *adapted state vector* at time t_1. Then, a random walk is performed, using the transition probabilities given by the adapted transition matrices until the final observation of o at time t_m is reached: For each $t_1 < t \leq t_m$ a state $P(o,t)$ is sampled by a random transition using the adapted transition matrix $F^o(t_1)$. Thus, $P(o,t)$ is the realization of the random variable defined by the $P(o,t-1)$'th line of the adapted transition matrix $F^o(t)$. On these certain trajectories $P(o,t), t_1 \leq t \leq t_{|\Theta^o|}$, which are realized for each object, a standard LCSS algorithm[28] for certain trajectories can be applied.

Lemma 14.2. The relative frequency of the $LCSS_{\epsilon,\delta}(o_1, o_2)$ values yields an un-biased estimator of the distribution of the uncertain $ULCSS_{\epsilon,\delta}(o_1, o_2)$ given observations and transition matrices of objects o_1 and o_2.

Proof. The random variable $ULCSS_{\epsilon,\delta}(o_1, o_2)$ follows a multinomial distribution. Thus, the random variable $ULCSS_{\epsilon,\delta}(o_1, o_2) = k$ that the LCSS of two uncertain spatio-temporal objects equals some integer k follows a binomial distribution. This way, we can deduce a confidence region of the $ULCSS_{\epsilon,\delta}(o_1, o_2)$-vector as a product of these intervals. Proceeding that way allows to use all the existing methods for the binomial distribution. In particular, we can use Chebyshev inequality to bound the maximum error for each $P(ULCSS = k)$:

$$P(|\hat{p} - p| \geq \epsilon) \leq \frac{p \cdot (1 - p)}{\epsilon^2 \cdot n} \leq \frac{1}{4\epsilon^2 \cdot n}$$

.

Clearly, the approximation for each probability approaches zero if the number of samples approaches infinity, thus the sum of error also approaches zero. However, it is difficult to take into account the constraint that the components of p sum up to 1. Thus, the resulting bounds will be looser than necessary. More advanced techniques to bound the approximation error that are able to take this constraint into account, are described in [43]. □

14.6 Experimental Evaluation

All tested methods were implemented in C++. For performing the sparse matrix operations we utilized the Eigen[7] package. All experiments were run on a single 64-Bit machine with an Intel Core i7 processor with 2.93 GHz and 8GB of RAM.

14.6.1 Datasets and Experimental Setting

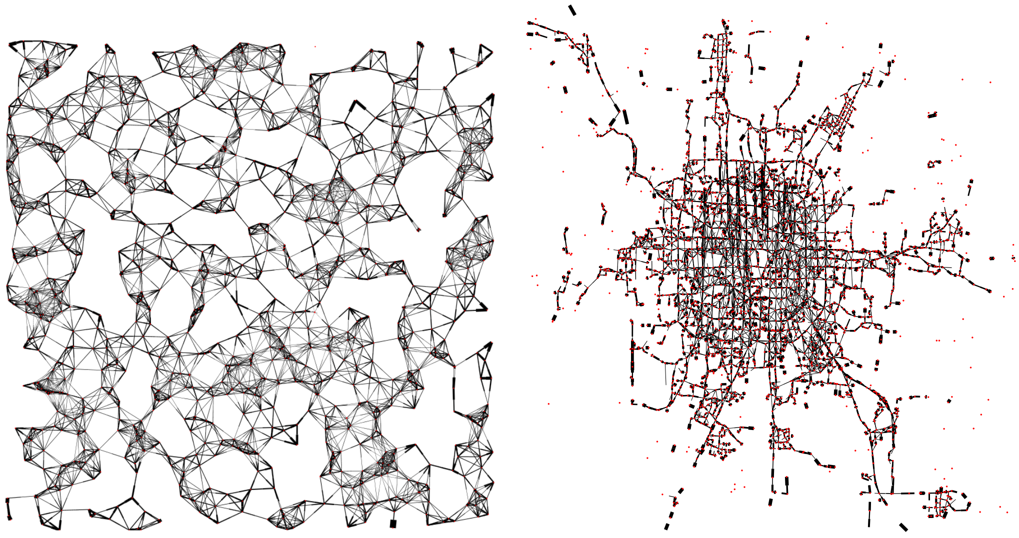
Our evaluation focuses on evaluating the computation of ULCSS (uncertain versions of other similarity measures, like DTW, is beyond the scope of this chapter). For the sampling-based approach both the time for constructing a sampling object (performed each time an object is added to the database) and the performance of the actual sampling have to be taken into account. While the exact ULCSS computation is fixed to $\delta = 0$, we ran the experiments of the sampling based approach with both $\delta = 0$ (UALCSS) and $\delta = 5$ (ULCSS) in order to assess the exact computation and to show the impact of higher values for δ in approximate computation.

Artificial Data

The experiments on artificial data are based on a state space in the two-dimensional Euclidean space, consisting of n states. Each of these states is drawn uniformly from the

---

[7] http://eigen.tuxfamily.org/index.php



(a) The synthetic dataset

(b) The real dataset

Figure 14.6: Examples of the state space (nodes) and the transition probabilities (edges) between them.

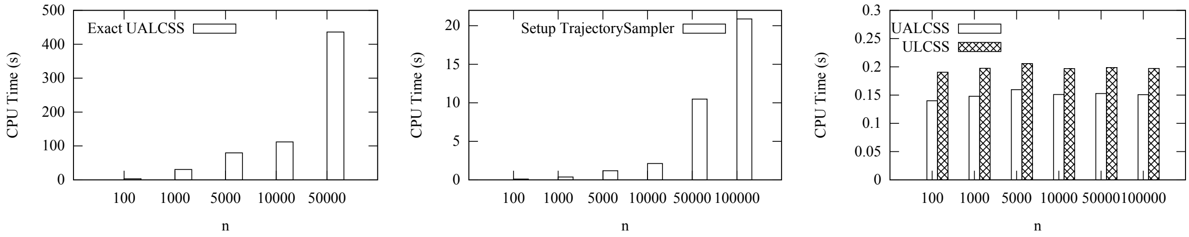


Figure 14.7: Experimental results - varying n

$[0,1]^2$ space. Afterwards, a graph was created from these states by connecting points with an Euclidean distance smaller than $r = \sqrt{\frac{b}{n*\pi}}$, with b being the average number of neighbours of a state. This choice of r ensures that the number of neighbors of a state does not depend on the number of states in general, which is a straightforward assumption in spatio-temporal databases (e.g., the number of lanes exiting a crossing does not depend on the number of crossings in general).

The graph's edge weights were assigned indirectly proportionally to the distance of a state to its neighbour, assuming that it is more probable that during a transition an object moves to a closer state than to a state further away. The marked adjacency matrix of the resulting graph was then used as an object's transition matrix in the following. An example graph constructed by this method is shown in Figure 14.6(a). States are drawn as circles and connections between states with a non-zero transition probability are drawn as black lines.

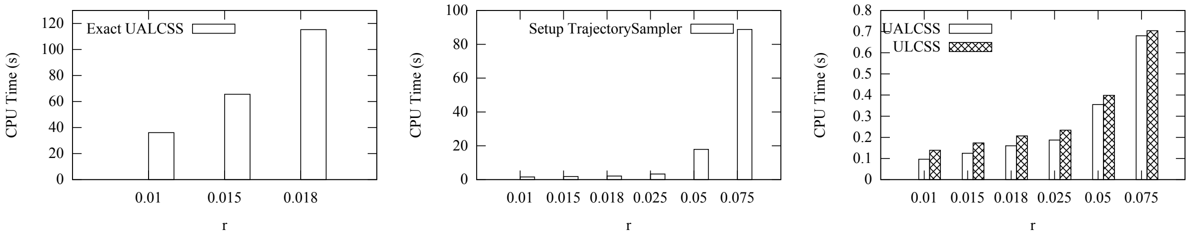


Figure 14.8: Experimental results - varying r

Variable	Values	Unit
n	100, …, **10000**, …, 100000	states
l	25, …, **50**, …, 125	timesteps
i	5, …, **10**, …, 50	timesteps
r	0.01,…, $\sqrt{\frac{b}{n*\pi}}$, …, 0.075	space units
δ	0,…, **5**, …, 50	timesteps
δ	5000, …, **10000**, …, 1000000	samples

Table 14.1: Values for the used independent variables (synthetic data)

Real Data

In addition to experiments on artificial data we performed experiments on a street network dataset from the city of Beijing, created by the authors of [66]. The dataset consists of 3008 states, the crossings of a street network. These are connected according to the turning probabilities of cars extracted from a GPS dataset. Figure 14.6(b) visualizes this real dataset. Based on this transition matrix, a random trajectory was drawn to construct (certain) observations of an uncertain object, and every i-th point from this trajectory was used as an observation of the uncertain object.

In the experimental evaluation we varied the number of states n, the length l of the analyzed time interval, the interval i between two observations, the range r, and the allowed amount of time shifting δ to measure the runtime of computing the ULCSS. Additionally we varied the number of samples σ in order to measure the quality of the resulting approximated distribution. The values used for these variables used in the experiments can be found in Table 14.1; default values are denoted in bold.

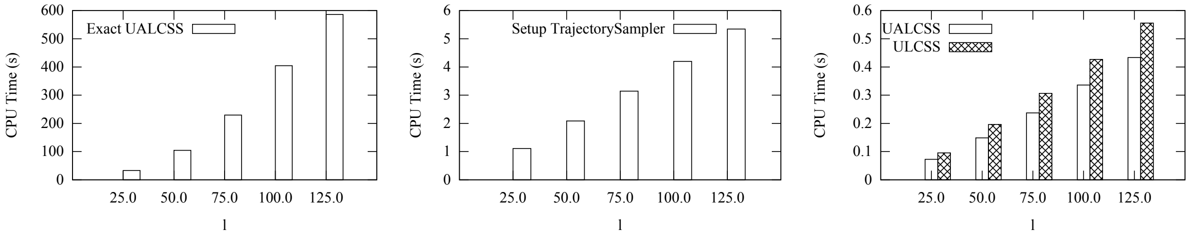


Figure 14.9: Experimental results - varying l

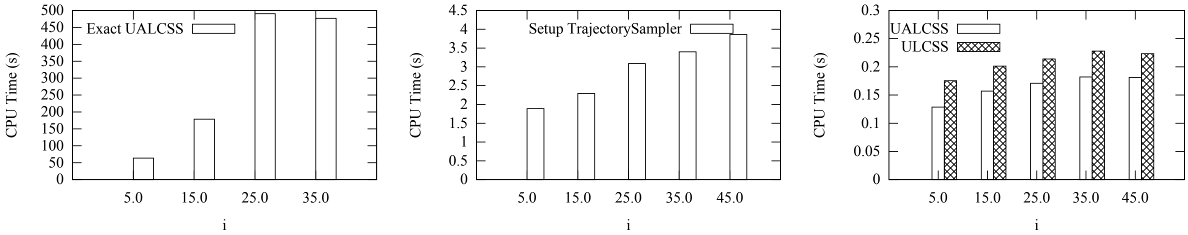


Figure 14.10: Experimental results - varying i

14.6.2 Results

Varying the Number of States n.

Constructing the underlying state network with a higher resolution or increasing the size of the world both increase the number of states in the network. Within this experiment we aimed at varying the worlds *size* (n), while increasing r (the next experiment) can be interpreted as increasing the *resolution* of a world.

As shown in Figure 14.8, with increasing n, matrix operations become more costly such that both the performance of the exact ULCSS computation and the performance of constructing the simplified transition matrices for the trajectory sampler drops. The number of non-zero entries in the matrix array of the UALCSS algorithm grows faster than the matrices stored in the trajectory sampler. Let $|P(o,t)|$ denote the number of states where o might be located with non-zero probability. Then a matrix computed by the exact algorithm contains $|P(o_1,t)| * |P(o_2,t)|$ non-zero entries. For the trajectory sampler, a matrix contains about $|P(o| * const$ non-zero entries.

Note that the actual sampling process of the suggested approximation algorithm does not depend on n, such that whenever a sampling object has been computed once, the size of the underlying state space barely affects the performance once the simplified transition matrices have been computed.

Varying the Range r of Connectivity.

Varying the range of connectivity r clearly shows a negative impact on the performance of all approaches. With a higher connectivity, the filling degree of transition matrices increases, such that more states can be reached in a shorter amount of time. The higher filling degree of the used matrices decreases the positive impact of using sparse instead of dense matrix arithmetic.

Varying the Length l of the LCSS.

Increasing the length of the time interval for which the ULCSS has to be computed (Figure 14.9) increases the number of iterations for the exact approach such that more matrix multiplications have to be performed during each iteration. Note that the number of matrix multiplications for this algorithm is $O(t^2)$; i.e., the overal runtime complexity is superlinear

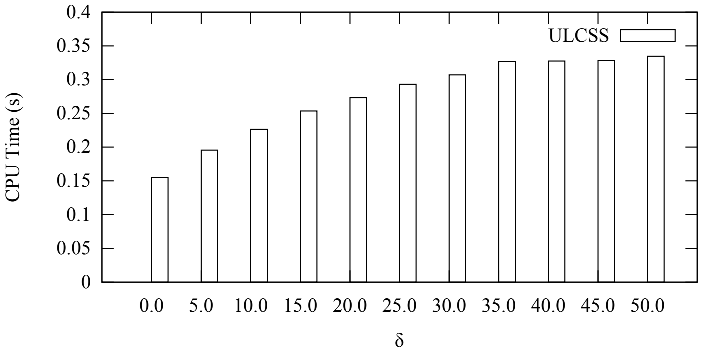


Figure 14.11: Experimental results - varying δ

in t. In contrast, the number of matrix multiplications performed when sampling is in $O(t)$, which is another advantage of sampling. Since the computation of the UALCSS is linear in t for the sampling-based approach, the remaining sampling process is efficient as well.

Varying the Interval i between Observations.

When increasing the time interval between two observations (Figure 14.10), objects become more uncertain, i.e., the objects can be located in more states. Therefore state vectors, simplified transitioning matrices, and the matrices generated by the exact UALCSS algorithm contain more entries, leading to a worse runtime performance of all algorithms. While the negative effect on the exact approach is quite large, the impact on the sampling process is much lower. Furthermore note that the actual sampling process of drawing trajectories for an uncertain object increases as well, because if there are only a few observations, an object can move along more paths than with a higher number of observations.

Varying the Degree of Time Shifting (δ).

The degree of time shifting (δ) can only be set on the approximate approach. It is well-known that the LCSS problem can be solved in $O(t * \delta)$ time. The experiments shown in Figure 14.11 show a similar behavior. Note that the runtime for the LCSS computation converges for higher values of δ. In our experiments, the default length of a sequence was set to 50. Therefore, according to the definition of the LCSS, for $\delta = 25$ nearly every element of the first sequence has to be compared to nearly all elements of the second sequence, which is only slightly better than for $\delta = 50$.

Sampling Quality: Varying the Number of Samples σ.

To evaluate the approximation quality of the sampling-based approach we computed the sum of differences between buckets from the exact and approximate distributions; the results can be found in Figure 14.12. With 10.000 samples, the approximate distribution deviates from the exact distribution by about 2.5 percent. Higher accuracy can be achieved by drawing a larger number of samples.

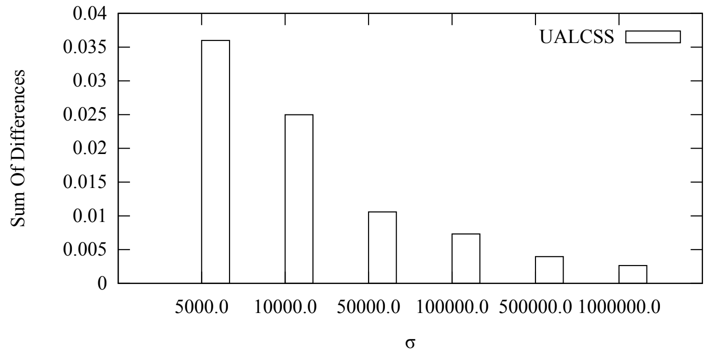


Figure 14.12: Approximation quality - varying σ

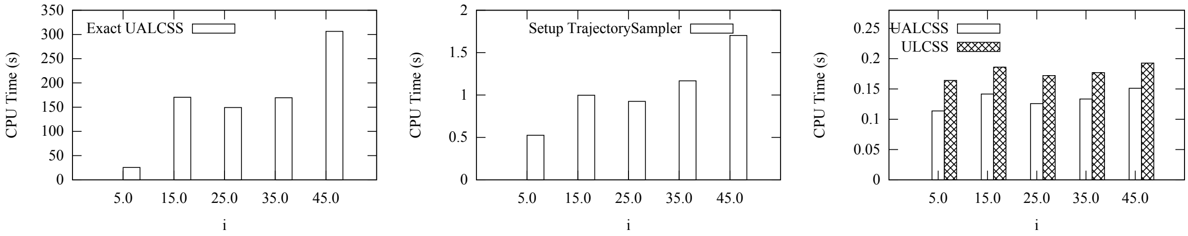


Figure 14.13: Experimental results (real data), varying i

Real Data Experiments.

A snapshot of experiments on real data can be found in Figure 14.13, which visualizes the impact of the interval between observations on the runtime of both the exact and the sampling-based LCSS computation. The resulting curves fit nicely to the ones from the artificial dataset. The results when varying other parameters are similar to the corresponding ones on synthetic data and they are omitted due to space constraints.

14.7 Sampling for arbitrary Queries

Clearly, this sampling approach can be used to solve arbitrary queries in an uncertain setting based on the Markov model. The general idea of using the sampling approach is the following:

1. Compute the adapted model for each object in the database
2. Until the desired accuracy is reached:
    (a) Sample a trajectory for each of these objects using this adapted model
    (b) Solve the query in this certain setting
    (c) Increase a counter defined by the outcome of the query
3. Return query results based on the resulting distribution

Obviously, additional optimizations, pruning techniques and index structures can be applied to this basic approach, increasing the speed of query evaluation. For example, during nearest neighbor query processing, pruning can be employed in order to avoid sampling of objects too far away from a query object.

14.8 Conclusion

In this chapter, we addressed the problem of similarity search on uncertain spatio-temporal data under the Markov-Chain-Model. We developed an exact algorithm to compute the LCSS distribution of two uncertain objects. While the complexity of this exact approach is polynomial, it can only be employed to settings without time shifting. Furthermore, the complexity of $O(\Delta t^2 \mathcal{S}^3)$ is still prohibitively large.

To tackle both of these problems, we proposed a sampling-based approach that uses Bayesian inference to adapt the transition matrices of the underlying uncertain objects, enabling the use of Monte-Carlo based sampling techniques. This does not only lead to a lower runtime complexity of LCSS computation, but also allows to approximate the LCSS with arbitrary time shifting δ.

In our experimental analysis we have shown that the theoretical runtime complexity of both the exact and approximate solutions also hold in practical situations. While the performance of computing the exact LCSS distribution between two objects is usually slow, sampling is generally much more feasible, even if the number of samples is high.

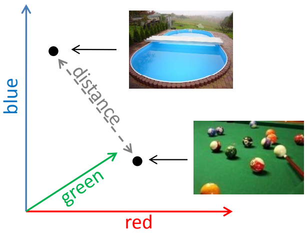
Last but not least, after adapting the transition matrices of an uncertain object, sampling can be used to solve arbitrary problems in an uncertain setting under the Markov-chain model, for example nearest neighbor queries or computing other similarity measures such as the Dynamic Time Warping Distance or the Edit Distance.

Chapter 15

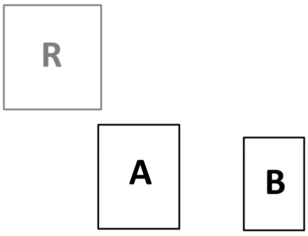
Final Remarks

In this chapter we conclude the thesis by summarizing and reviewing the contributions and giving ideas for future research on the covered topics.

15.1 Summary and Contributions



The main focus of this thesis is similarity search on three important data types, namely spatial data, uncertain spatial data and uncertain spatio-temporal data. The introductive first part of this work reviews the concept of feature based similarity search using feature vectors as prevalent representation of multi-attribute and geo-spatial objects in similarity search applications. Additionally, this part explicates common similarity query types (such as ϵ-range, k-nearest neighbor and reverse-k-nearest neighbor queries) and efficient processing techniques (such as spatial index structures and multi-step query processing).



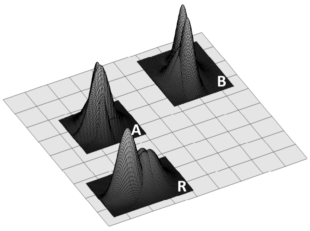
The second part of this work concentrates on spatial data and introduced the concept of *spatial domination* as module to boost the introduced similarity queries when spatial approximations are involved. To detect if "object A is definitely closer to object R than object B" (given these objects by approximations) it is necessary to develop domination decision criteria (DDC). State-of-the-art techniques for detecting *spatial domination* suffer from the problem of incompleteness (*MinMaxDist* DDC) or restrictions regarding the approximations (The *Hyperplane* DDC requires A and B to be a point). The main problem here are so called *distance dependencies.* Starting from the *Hyperplane* domination decision criterion, which is a geometrical approach, successively new criteria were developed in order to mitigate these restrictions:

- The *EntryHyperplane* DDC removes the restriction for A to be a point.
- The *CornerBased* DDC removes the restriction for B to be a point, still yielding exponential runtime.

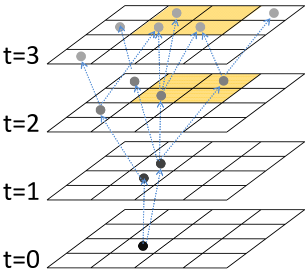
- The *Trigonometric* DDC is also efficiently computable but requires spherical approximations and Euclidean space.

- The *Optimal* DDC is complete, correct and efficiently computable for rectangular approximations under all L_p-norms.

Based on the definition of *spatial domination* we introduced the module of *domination count* which in the most basic form is approximated by naively counting the number of objects $A_i \in \mathcal{D}$ in the database which dominate an object B w.r.t. a third object R. However we showed that the approximation based on the basic *domination count* can be improved by considering *partial domination*. The efficiency and the wide applicability of the proposed methods was experimentally shown within the scope of various applications and query predicates.



The third part focused on uncertain spatial data. Uncertainty is an inherent characteristic of spatial data in many scenarios (due to inaccurate, missing or contradictory values of attributes). It introduces another layer of complexity and cannot not just be treated as a simple extension to spatial data. As similarity queries are as important on uncertain spatial data as on pure spatial data we extended the definition of *spatial domination* to *probabilistic spatial domination*. We showed how the probability of the event “object A is definitely closer to object R than object B” can be approximated in accordance with the possible worlds semantics avoiding costly integration operations. Next we extended the *domination count* to *probabilistic domination count* and revealed the problem of setting off the results from the single *probabilistic spatial domination* computations against each other. Our solution to this problem is a sophisticated method based on the well-known generating functions which we name *uncertain generating functions*. Based on this technique it is possible to approximate the *probabilistic domination count*. We showed how this can be used in order to delay costly integrations during query processing as long as possible. Plugged into the application of probabilistic reverse nearest neighbor query processing, our new technique is able to outperform state-of-the art algorithms for this problem.



In the fourth part of this thesis the dimension of time is included in the data and thus this part focused on uncertain spatio-temporal data. There is a body of work on this subject, which bounds the possible movement of objects by spatial approximations allowing for simple queries like “Is it possible that an object passed through a query region within some time interval?”. However none of the existing works so far was able to assign a confidence to this event according to possible worlds semantics. The main problem here is that prevalent methods ignored the *location dependencies* between positions of an object at successive points of time. We investigated the possibility to model the movement of objects by stochastic processes, in particular Markov Chains. This has three major advantages: (1) It is now possible to assign probabilities to query results, (2) the assigned probabilities are in accordance with possible worlds semantics and (3) computations are based on sparse matrix multiplications for which there exist extremely efficient methods. Based on this model we were able to find efficient solutions for three spatio-temporal window query predicates ($\forall, \exists$ and $k - times$). In order to allow querying of large databases we additionally developed the UST-Index for the organization of uncertain spatio-temporal data sets. The main advantage over existing index structures for spatio-temporal data is that the UST-Index implements the concept of probabilistic pruning in addition to spatial pruning, i.e. it is possible to prune objects based on the probability threshold τ of a query. This is possible through precomputing the probability of each object to stay within a spatio-temporal approximation with probabilistic guarantees called sub-diamond (of the original diamond between two observations) and to bound this probability using an optimized linear function. The superiority of the UST-Index over the R*-Tree in terms of CPU-cost and I/P-cost was experimentally shown. Last but not least we showed that there exist similarity queries (e.g. based on ULCSS) which yield exponential runtime when modeling UST data with Markov-Chains. Our solution to the problem was a sampling approach which adapts the transition matrices of each object using Bayesian inference. Using this method it is possible to approximately answer arbitrary queries on UST data in an efficient manner.

15.2 Future Work

We believe that the subjects covered in this thesis not only yield sophisticated solutions for several problems but also serve as a starting point for research in several directions. Specifically we want to highlight the following ideas for future work.

Regarding the concepts of *spatial domination* and *domination count* we plan to investigate the applicability of the proposed criteria to other query scenarios. In fact we already applied the techniques to RkNN monitoring [61], inverse queries [29] and RkNN join processing [67]. However there are numerous applications which can be enhanced using the two proposed concepts. Additionally, we believe that data mining tasks such as clustering and outlier detection on datasets with extended object approximations might be an interesting field to consider. Another direction would be to extend the *Optimal* domination decision criterion to other types of approximations (e.g. ellipses or general polyhedrons) and distance functions (e.g. weighted L_p-Norms, quadratic forms or earth mover's distance). This could be achieved by regarding the spatial domination problem as an optimization problem (which is possible for the case of rectangles). Last but not least the transformation of the ideas to non-vector spaces remains an open problem. However the definition *spatial domination* can be straightforwardly transferred to metric spaces in general. The problem of *distance dependencies* remains in these spaces. We plan to investigate *reference point embeddings* of datasets for query processing and are confident that techniques can be found to get a grip of the *distance dependencies* in this setting.

A similar perspective holds for the methods covered in Part 3 of this thesis. The techniques for *probabilist spatial domination* and *probabilist domination count* computation can be applied to a larger set of applications. We started by utilizing them in the context of Voronoi cell computation on uncertain spatial data [177]. The technique of *uncertain generating functions* is also of a more general nature and can not only be used in the context of uncertain spatial data. We also plan to use this technique in other scenarios such as aggregation queries in uncertain relational databases.

Regarding Part 4 of this thesis we believe that the described parts (model, index structure and sampling techniques) are the basis of a general framework for managing uncertain spatio-temporal data. From here there are several new interesting research directions to follow. The first idea is to investigate other spatio-temporal query types than the ones considered so far (spatio-temporal window queries and similarity queries based on ULCSS) and search for efficient solutions based on the existing model. Though the Markov Chain model has shown to be a useful choice for representing probabilistic spatio-temporal data there is still much space for improvement to yield a prototype usable for real-life applications. Future work should try to relax the assumptions on which the model is based:

- The assumption of discrete space – which is only applicable in some applications, where the universe of discuss is finite (e.g. a road network with a finite number of edges and vertices) or in applications where a discretization of space (e.g. by a 2D or 3D grid) does not incur too much loss of information.
- The assumption of discrete time – which rarely holds in real applications, where time cannot be abstracted to a series of uni-distant points in time (known as ticks). Making this assumption may yield incorrect results, by missing events in-between ticks.
- The assumption that objects are mutually independent – which may not hold in practice where additional information like "person A and person B travel together" may be given. In this case, the positions of A and B will be highly correlated.
- The assumption of the model being given and correct – which is the strongest assumption. So far no techniques were presented to learn the parameters of the Markov chain. Thus a more in-depth evaluation of existing techniques to learn models, as well as the invention of new learning methods is required.

Developing new models and query processing techniques based on more powerful stochastic processes like Markov Processes, Continuous Markov Chains or Wiener Processes is in our opinion a challenging but profitable way to go.

In addition to querying spatio-temporal data, knowledge discovery in spatio-temporal data, in particular trajectory clustering, trajectory pattern mining and spatio-temporal collocation mining (i.e. the problem of finding sets of spatially and temporally correlated events) has become a hot research topic in the recent years. Many works exist for mining certain trajectories (for an overview see [104, 180]). The few existing works on uncertain trajectory mining (e.g. [164, 130]) usually simplify the problem and do not return results according to the possible worlds semantics since the underlying model is not powerful enough to allow for considering *temporal dependencies*. We believe that the last part of this work is the first step for a large framework for managing and mining uncertain spatio-temporal data [1] and a starting point for challenging but thrilling research work.

---

[1] A framework including all mentioned techniques is currently under development and near completion (http://www.dbs.ifi.lmu.de/cms/Publications/UncertainSpatioTemporal)

List of Figures

2.1 The KDD Process 8
2.2 (Dis-)similarity between two images in the feature space 9
2.3 Example of a range query with q as query object 11
2.4 Example of a 5NN query with q as query object 12
2.5 The two types of RkNN queries 13
2.6 An R-Tree for 2D points 15
2.7 Multi-step query processing 16
2.8 TPL pruning ($k = 1$). 17

3.1 Approximations for complex polygons 22
3.2 MBR pruning example 23
3.3 a_1 and a_2 are collectively dominating R w.r.t. b 26
3.4 Domination for pruning 28

4.1 The *EntryHyperplane* criterion 32
4.2 Construction of the hyperplane H_{Ab} 33
4.3 Partial pruning based ranking count estimation strategy. 34
4.4 Example to Lemma 4.3. 35
4.5 Hyperplane decision criterion ($k = 1$). 38
4.6 Conservative approximation $H_{e'q}$ of the hyperplanes associated with all objects of an index entry e'. 39
4.7 Partial pruning for ranking count computation. 40
4.8 Illustration of entries that are/are not affected by the refinement of an entry e. 41
4.9 Comparison of our RkNN ranking with the competitors on uniformly distributed data. 44
4.10 Comparison of our RkNN ranking with the competitors on clustered data. 45
4.11 Comparison of our RkNN ranking and the competitors on 5D gene expression data. 46
4.12 Effect of the spatial partial pruning. 47

5.1 Geometric interpretation of Dom(A, B, R) 50
5.2 Partitioning of objects A and B for the 2d case. 52
5.3 R1NN examples. 55
5.4 R1NN selectivity estimation based on position approximations. 57
5.5 Conservative and progressive pruning. 59
5.6 Different pruning potentials. 61
5.7 MMP vs. CBP on uniform dataset 64
5.8 MMH vs. MPPH on uniform and tac dataset 65
5.9 Impact of parameter m on the number of refinements 66
5.10 MMH vs. MPPH with varying size of the two sets 66

6.1 Domination on spheres 70
6.2 Considering the surface points of R is sufficient 72
6.3 Considering only points in P and R is sufficient 73
6.4 Trigonometric domination computation 74
6.5 Domination for AkNN query processing 80
6.6 The gray shaded area shows the active region used by trigonometric pruning (see Definition 6.8) which results from the intersection of the three pruning areas of P_1, P_2, P_3. The dashed line $[P1, P4]$ indicates the GlobalDist$(B_{\mathcal{R}}, S)$ which defines the pruning area (dashed outer circle) used by BNN in case of a 1NN query. 85
6.7 Comparison of MBA, XBNN and TP for $k = 10$. The figure displays the sum of I/O and CPU times for one AkNN query. 87
6.8 I/O (left) and CPU (right) experiments for AkNN queries on different data sets by all algorithms 88
6.9 ϵ of TP for an A-10-NN query on FC. 89
6.10 CPU-times on different data sets. 89
6.11 Grouping procedures of the outer set, validated via XBNN. 90
6.12 A-10-NN queries on clustered data sets of varying dimensions 90

7.1 Illustration of Lemma 7.2. 97
7.2 Partial domination using grid partitioning 101
7.3 Domination Count estimation using grid partitioning. 102
7.4 Example for computing DC_{bisect} 103
7.5 Refinement areas of DDC$_{MM}$ and DDC$_{OPT}$ 104
7.6 Comparison of MinMax- and optimal-criterion on synthetic data 106
7.7 Heuristics for partial domination 106

7.8 *AKKRZ* using different decision criteria. Page accesses (top row) and distance calculations (bottom row). 108
7.9 Evaluation of the different decision criteria for 10 nearest neighbor queries. 109

8.1 Variants of attribute uncertainty 116
8.2 Variants of uncertain objects 118
8.3 A dominates B w.r.t. R with high probability. 120
8.4 Example of probabilistic domination 122

9.1 Probabilistic domination 126
9.2 Approximated PDF of $\sum_{i=1}^{2} X_i$. 132
9.3 Runtime of **MC** for increasing sample size. 141
9.4 Optimal vs. MinMax decision criterion. 142
9.5 Uncertainty of **IDCA** w.r.t. the relative runtime to **MC**. 142
9.6 Runtimes of **IDCA** and **MC** for different query predicates k and τ. 143
9.7 Impact of influencing objects. 144

10.1 Uncertain object example: user ratings. 148
10.2 Examples for RNN and PRNN. 150
10.3 Pruning uncertain objects using minimal and maximal distance. 152
10.4 Comparison of different pruning techniques. 163
10.5 Performance of the PRNN Algorithm. 164
10.6 Behaviour of the PRkNN-Algorithm. 166

11.1 (Uncertain) Spatio-Temporal Data 170
11.2 Models for UST data 171
11.3 location dependencies in UST data 174

12.1 Querying Uncertain Spatio-Temporal Data 178
12.2 Some possible worlds of one uncertain object 181
12.3 Schematic illustration of object-based (OB) and query-based (QB) query processing. 183
12.4 Procedure of object-based (OB) and query-based (QB) query processing. . 184
12.5 Multiple observations of an object 188
12.6 Two observations of an object 190
12.7 Query processing runtime w.r.t. the number of states. 195
12.8 Query processing runtime w.r.t. the size of the query time frame 195
12.9 Performance evaluation of query predicates. 196
12.10 Comparison of QB and OB behavior with scaling parameters 196

13.1 Spatio-temporal data. 199
13.2 Spatio-temporal approximation. 203
13.3 Intersection between query and diamond 205
13.4 Probabilistic diamonds 206
13.5 The UST-tree. 214
13.6 Diamond construction 217
13.7 Overall performance (synthetic data set) 218
13.8 Experiments on synthetic data 220
13.9 Experiments on real data (∀) 221

14.1 Uncertain trajectories. 225
14.2 The possible paths and possible worlds $\{w_{1-4}\}$ of two uncertain objects. Possible worlds have a probability of 0.25 each. 229
14.3 Traditional MC-sampling. 233
14.4 An overview over our forward-backward-algorithm. 235
14.5 Exemplary Markov Chain, Visualization (a) and Transition Matrix (b). . . 241
14.6 Examples of the state space (nodes) and the transition probabilities (edges) between them. 244
14.7 Experimental results - varying n 244
14.8 Experimental results - varying r 245
14.9 Experimental results - varying l 245
14.10 Experimental results - varying i 246
14.11 Experimental results - varying δ 247
14.12 Approximation quality - varying σ 248
14.13 Experimental results (real data), varying i 248

List of Tables

3.1 Overview over spatial domination decision criteria 30

6.1 Table of notations 79

8.1 Tuple uncertainty example 115
8.2 Attribute uncertainty example 115
8.3 Sensor readings 116
8.4 Possible worlds 117

10.1 Parameters and their default values. 162

12.1 Table of notations 182
12.2 Parameters for the synthetic datasets 194

14.1 Values for the used independent variables (synthetic data) 245

Bibliography

[1] Abiteboul, S., Kanellakis, P. C., and Grahne, G. On the representation and querying of sets of possible worlds. *Theoretical Computer Science 78*, 1 (1991), 158–187.

[2] Achtert, E., Böhm, C., Kröger, P., Kunath, P., Pryakhin, A., and Renz, M. Approximate reverse k-nearest neighbor queries in general metric spaces. In *Proceedings of the 15th ACM Conference on Information and Knowledge Management (CIKM), Arlington, VA* (2006), pp. 788–789.

[3] Achtert, E., Böhm, C., Kröger, P., Kunath, P., Pryakhin, A., and Renz, M. Efficient reverse k-nearest neighbor search in arbitrary metric spaces. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Chicago, IL* (2006), pp. 515–526.

[4] Achtert, E., Kriegel, H.-P., Kröger, P., Renz, M., and Züfle, A. Reverse k-nearest neighbor search in dynamic and general metric databases. In *Proceedings of the 12th International Conference on Extending Database Technology (EDBT), Saint-Petersburg, Russia* (2009), pp. 886–897.

[5] Achtert, E., Kriegel, H.-P., and Zimek, A. ELKI: a software system for evaluation of subspace clustering algorithms. In *Proceedings of the 20th International Conference on Scientific and Statistical Database Management (SSDBM), Hong Kong, China* (2008), pp. 580–585.

[6] Agrawal, P., Benjelloun, O., Sarma, A. D., Hayworth, C., Nabar, S., Sugihara, T., and Widom, J. Trio: A system for data, uncertainty, and lineage. In *Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB), Seoul, Korea* (2006).

[7] Agrawal, R., Faloutsos, C., and Swami, A. Efficient similarity search in sequence databases. In *Proceedings of the 4th International Conference on Foundations of Data Organization and Algorithms (FODO), Chicago, IL* (1993), pp. 69–84.

[8] Amengual, J. C., and Vidal, E. Efficient error-correcting viterbi parsing. *IEEE Transactions on Pattern Analysis and Machine Intelligence 20* (1998), 1109–1116.

[9] Ankerst, M., Breunig, M. M., Kriegel, H.-P., and Sander, J. OPTICS: Ordering points to identify the clustering structure. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Philadelphia, PA* (1999), pp. 49–60.

[10] Antova, L., Jansen, T., Koch, C., and Olteanu, D. Fast and simple relational processing of uncertain data. In *Proceedings of the 24th International Conference on Data Engineering (ICDE), Cancun, Mexico* (2008).

[11] Arratia, R., and Waterman, M. S. An erdös-renyi law with shifts. *Advances in mathematics 55*, 1 (1985), 13–23.

[12] Ashbrook, D., and Starner, T. Using gps to learn significant locations and predict movement across multiple users. *Personal and Ubiquitous Computing 7* (2003), 275–286.

[13] Asuncion, A., and Newman, D. J. UCI Machine Learning Repository, 2007. `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

[14] Assfalg, J., Kriegel, H.-P., Kröger, P., Kunath, P., Pryakhin, A., and Renz, M. Similarity search on time series based on threshold queries. In *Proceedings of the 10th International Conference on Extending Database Technology (EDBT), Munich, Germany* (2006), pp. 276–294.

[15] Assfalg, J., Kriegel, H.-P., Kröger, P., and Renz, M. Probabilistic similarity search for uncertain time series. In *Proceedings of the 21st International Conference on Scientific and Statistical Database Management (SSDBM), New Orleans, LA* (2009), pp. 435–443.

[16] Bacchus, F., Grove, A. J., Halpern, J. Y., and Koller, D. From statistical knowledge bases to degrees of belief. *Artificial Intelligence 87*, 1-2 (1996), 75–143.

[17] Bâdoiu, M., and Clarkson, K. L. Smaller core-sets for balls. In *SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms* (Philadelphia, PA, USA, 2003), Society for Industrial and Applied Mathematics, pp. 801–802.

[18] Barbará, D., Garcia-Molina, H., and Porter, D. The management of probabilistic data. *IEEE Transactions on Knowledge and Data Engineering 4*, 5 (1992), 487–502.

[19] Bast, H., Majumdar, D., Schenkel, R., Theobald, M., and Weikum, G. Io-top-k: Index-access optimized top-k query processing. In *Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB), Seoul, Korea* (2006), pp. 475–486.

[20] Bayer, R., and McCreight, E. M. Organization and maintenance of large ordered indices. *Acta Informatica 1* (1972), 173–189.

[21] Beckmann, N., Kriegel, H.-P., Schneider, R., and Seeger, B. The R*-Tree: An efficient and robust access method for points and rectangles. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Atlantic City, NJ* (1990), pp. 322–331.

[22] Bellman, R. *Adaptive Control Processes. A Guided Tour.* Princeton University Press, 1961.

[23] Benetis, R., Jensen, C. S., Karčiauskas, G., and Šaltenis, S. Nearest and reverse nearest neighbor queries for moving objects. *The VLDB Journal 15*, 3 (2006), 229–249.

[24] Bentley, J. L. Multidimensional binary search trees used for associative searching. *Commun. ACM 18*, 9 (1975), 509–517.

[25] Berchtold, S., Böhm, C., Jagadish, H. V., Kriegel, H.-P., and Sander, J. Independent Quantization: An index compression technique for high-dimensional data spaces. In *Proceedings of the 16th International Conference on Data Engineering (ICDE), San Diego, CA* (2000), pp. 577–588.

[26] Berchtold, S., Keim, D. A., and Kriegel, H.-P. The X-Tree: An index structure for high-dimensional data. In *Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB), Bombay, India* (1996), pp. 28–39.

[27] Berchtold, S., and Kriegel, H.-P. S3: similarity search in cad database systems. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Tucson, AZ* (1997), pp. 564–567.

[28] Bergroth, L., Hakonen, H., and Raita, T. A survey of longest common subsequence algorithms. In *Proceedings of the 7th International Symposium on String Processing and Information Retrieval (SPIRE)* (2000), pp. 39–48.

[29] Bernecker, T., Emrich, T., Kriegel, H.-P., Mamoulis, N., Renz, M., Zhang, S., and Züfle, A. Inverse queries for multidimensional spaces. In *Proceedings of the 12th International Symposium on Spatial and Temporal Databases (SSTD), Minneapolis, MN* (2011), pp. 330–347.

[30] Bernecker, T., Emrich, T., Kriegel, H.-P., Mamoulis, N., Renz, M., and Züfle, A. A novel probabilistic pruning approach to speed up similarity queries in uncertain databases. In *Proceedings of the 27th International Conference on Data Engineering (ICDE), Hannover, Germany* (2011), pp. 339–350.

[31] Bernecker, T., Emrich, T., Kriegel, H.-P., Renz, M., Zankl, S., and Züfle, A. Efficient probabilistic reverse nearest neighbor query processing on uncertain data. In *Proceedings of the 37th International Conference on Very Large Data Bases (VLDB), Seattle, WA* (2011), pp. 669–680.

[32] Bernecker, T., Kriegel, H.-P., Mamoulis, N., Renz, M., and Züfle, A. Scalable probabilistic similarity ranking in uncertain databases. *IEEE Transactions on Knowledge and Data Engineering 22*, 9 (2010), 1234–1246.

[33] Bernecker, T., Kriegel, H.-P., and Renz, M. ProUD: probabilistic ranking in uncertain databases. In *Proceedings of the 20th International Conference on Scientific and Statistical Database Management (SSDBM), Hong Kong, China* (2008), pp. 558–565.

[34] Bernecker, T., Kriegel, H.-P., Renz, M., Verhein, F., and Züfle, A. Probabilistic frequent itemset mining in uncertain databases. In *Proceedings of the 15th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Paris, France* (2009), pp. 119–128.

[35] Beskales, G., Soliman, M. A., and IIyas, I. F. Efficient search for the top-k probable nearest neighbors in uncertain databases. *Proc. VLDB Endow. 1*, 1 (2008), 326–339.

[36] Blais, E., and Blanchette, M. Common substrings in random strings. In *Combinatorial Pattern Matching* (2006), Springer, pp. 129–140.

[37] Boulos, J., Dalvi, N. N., Mandhani, B., Mathur, S., Ré, C., and Suciu, D. Mystiq: a system for finding more answers by using probabilities. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Baltimore, ML* (2005), pp. 891–893.

[38] Breunig, M. M., Kriegel, H.-P., Ng, R., and Sander, J. LOF: Identifying density-based local outliers. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Dallas, TX* (2000), pp. 93–104.

[39] Böhm, C., and Krebs, F. The k-nearest neighbor join: Turbo charging the KDD process. *Knowledge and Information Systems (KAIS) 6*, 6 (2004), 728–749.

[40] Böhm, C., Pryakhin, A., and Schubert, M. The Gauss-tree: Efficient object identification of probabilistic feature vectors. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE), Atlanta, GA* (2006), p. 9.

[41] Böhm, C., Pryakhin, A., and Schubert, M. Probabilistic ranking queries on gaussians. In *Proceedings of the 18th International Conference on Scientific and Statistical Database Management (SSDBM), Vienna, Austria* (2006), pp. 169–178.

[42] Cavallo, R., and Pittarelli, M. The theory of probabilistic databases. In *Proceedings of the 13th International Conference on Very Large Data Bases (VLDB), Brighton, UK* (1987), pp. 71–81.

[43] Chafai, D., and Concordet, D. Confidence regions for the multinomial parameter with small sample size. *Journal of the American Statistical Association 104*, 487 (2009), 1071–1079.

[44] Chapman, G., Cleese, J., Gilliam, T., Idle, E., Jones, T., and Palin, M. Monty python's the life of brian, 1979.

[45] Cheema, M. A., Lin, X., Wang, W., Zhang, W., and Pei, J. Probabilistic reverse nearest neighbor queries on uncertain data. *IEEE Transactions on Knowledge and Data Engineering 22*, 4 (2010), 550–564.

[46] Chen, J., and Cheng, R. Efficient evaluation of imprecise location-dependent queries. In *Proceedings of the 23rd International Conference on Data Engineering (ICDE), Istanbul, Turkey* (2007), pp. 586–595.

[47] Chen, Y., and Patel, J. M. Efficient evaluation of all-nearest-neighbor queries. In *Proceedings of the 23rd International Conference on Data Engineering (ICDE), Istanbul, Turkey* (2007), pp. 1056–1065.

[48] Chen, Z., Shen, H. T., and Zhou, X. Discovering popular routes from trajectories. In *Proceedings of the 27th International Conference on Data Engineering (ICDE), Hannover, Germany* (2011), pp. 900–911.

[49] Cheng, R., Chen, J., Mokbel, M., and Chow, C. Probabilistic verifiers: Evaluating constrained nearest-neighbor queries over uncertain data. In *Proceedings of the 24th International Conference on Data Engineering (ICDE), Cancun, Mexico* (2008), pp. 973–982.

[50] Cheng, R., Chen, J., Mokbel, M., and Chow, C. Probabilistic verifiers: Evaluating constrained nearest-neighbor queries over uncertain data. In *Proceedings of the 24th International Conference on Data Engineering (ICDE), Cancun, Mexico* (2008), pp. 973–982.

[51] Cheng, R., Chen, L., Chen, J., and Xie, X. Evaluating probability threshold k-nearest-neighbor queries over uncertain data. In *Proceedings of the 12th International Conference on Extending Database Technology (EDBT), Saint-Petersburg, Russia* (2009), pp. 672–683.

[52] Cheng, R., Kalashnikov, D., and Prabhakar, S. Querying imprecise data in moving object environments. In *IEEE Transactions on Knowledge and Data Engineering* (2004), vol. 16, pp. 1112–1127.

[53] Cheng, R., Kalashnikov, D. V., and Prabhakar, S. Evaluating probabilistic queries over imprecise data. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), San Diego, CA* (2003), pp. 551–562.

[54] Cheng, R., Singh, S., and Prabhakar, S. U-DBMS: a database system for managing constantly-evolving data. In *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB), Trondheim, Norway* (2005), pp. 1271–1274.

[55] Cheng, R., Xia, Y., Prabhakar, S., Shah, R., and Vitter, J. S. Efficient indexing methods for probabilistic threshold queries over uncertain data. In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB), Toronto, Canada* (2004), pp. 876–887.

[56] Ciaccia, P., Patella, M., and Zezula, P. M-Tree: an efficient access method for similarity search in metric spaces. In *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB), Athens, Greece* (1997), pp. 426–435.

[57] Cormode, G., Li, F., and Yi, K. Semantics of ranking queries for probabilistic data and expected results. In *Proceedings of the 25th International Conference on Data Engineering (ICDE), Shanghai, China* (2009), pp. 305–316.

[58] Emrich, T., Graf, F., Kriegel, H.-P., Schubert, M., and Thoma, M. Optimizing all-nearest-neighbor queries with trigonometric pruning. In *Proceedings of the 22nd International Conference on Scientific and Statistical Database Management (SSDBM), Heidelberg, Germany* (2010), pp. 501–518.

[59] Emrich, T., Graf, F., Kriegel, H.-P., Schubert, M., Thoma, M., and Cavallaro, A. CT slice localization via instance-based regression. In *Proceedings of the SPIE Medical Imaging Conference 2010: Image Processing (SPIE), San Diego, CA* (2010), vol. 7623, p. 762320.

[60] Emrich, T., Kriegel, H.-P., Kröger, P., Renz, M., Senner, J., and Züfle, A. A visual evaluation framework for spatial pruning methods. In *Proceedings of the 12th International Symposium on Spatial and Temporal Databases (SSTD), Minneapolis, MN* (2011), pp. 507–511.

[61] Emrich, T., Kriegel, H.-P., Kröger, P., Renz, M., Xu, N., and Züfle, A. Reverse k-nearest neighbor monitoring on mobile objects. In *Proceedings of the 18th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM GIS), San Jose, CA* (2010), pp. 494–497.

[62] Emrich, T., Kriegel, H.-P., Kröger, P., Renz, M., and Züfle, A. Constrained reverse nearest neighbor search on mobile objects. In *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM GIS), Seattle, WA* (2009), pp. 197–206.

[63] Emrich, T., Kriegel, H.-P., Kröger, P., Renz, M., and Züfle, A. Incremental reverse nearest neighbor ranking in vector spaces. In *Proceedings of the 11th International Symposium on Spatial and Temporal Databases (SSTD), Aalborg, Denmark* (2009), pp. 265–282.

[64] Emrich, T., Kriegel, H.-P., Kröger, P., Renz, M., and Züfle, A. Boosting spatial pruning: On optimal pruning of MBRs. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Indianapolis, IN* (2010), pp. 39–50.

[65] Emrich, T., Kriegel, H.-P., Mamoulis, N., Renz, M., and Züfle, A. Indexing uncertain spatio-temporal data. In *Proceedings of the 21th ACM Conference on Information and Knowledge Management (CIKM), Maui, HI* (2012), pp. 395–404.

[66] Emrich, T., Kriegel, H.-P., Mamoulis, N., Renz, M., and Züfle, A. Querying uncertain spatio-temporal data. In *Proceedings of the 28th International Conference on Data Engineering (ICDE), Washington, DC* (2012), pp. 354–365.

[67] Emrich, T., Kröger, P., Niedermayer, J., Renz, M., and Züfle, A. A mutual-pruning approach for rknn join processing. In *Proceedings of the 13th GI-Fachtagung für Datenbanksysteme in Business, Technologie und Web (BTW), Magdeburg, Germany* (2013).

[68] Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd ACM International Conference on Knowledge Discovery and Data Mining (KDD), Portland, OR* (1996), pp. 226–231.

[69] Fagin, R., and Halpern, J. Y. Reasoning about knowledge and probability. *Journal of the ACM (JACM) 41*, 2 (1994), 340–367.

[70] Fayyad, U., Piatetsky-Shapiro, G., and Smyth, P. Knowledge discovery and data mining: Towards a unifying framework. In *Proceedings of the 2nd ACM International Conference on Knowledge Discovery and Data Mining (KDD), Portland, OR* (1996), pp. 82–88.

[71] Forney, G. D. The viterbi algorithm. *Proceedings of the IEEE 61*, 3 (Mar. 1973), 268–278.

[72] Fu, J. C., and Lou, W. Y. W. Distribution of the length of the longest commmon subsequence of two multi-state biological sequences. *Journal of Statistical Planning and Inference 138* (2008), 3605 – 3615.

[73] Fuhr, N., and Rölleke, T. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Transactions on Information Systems 15*, 1 (1997), 32–66.

[74] Gaede, V., and Günther, O. Multidimensional access methods. *ACM Computing Surveys 30*, 2 (1998), 170–231.

[75] Ge, X., and Smyth, P. Deformable markov model templates for time-series pattern matching. In *Proceedings of the 6th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Boston, MA* (2000), pp. 81–90.

[76] Güting, R. H., and Schneider, M. *Moving Objects Databases.* Morgan Kaufmann, 2005.

[77] Guttman, A. R-Trees: A dynamic index structure for spatial searching. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Boston, MA* (1984), pp. 47–57.

[78] Hadjieleftheriou, M., Kollios, G., Tsotras, J., and Gunopulos, D. Indexing spatiotemporal archives. *The VLDB Journal 15*, 2 (2006), 143–164.

[79] Hariharan, R., and Toyama, K. Project lachesis: parsing and modeling location histories. In *Geographic Information Science* (2004), pp. 106–124.

[80] Hettich, S., and Bay, S. D. The UCI KDD archive. http://kdd.ics.uci.edu, 1999.

[81] Hey, T., Tansley, S., and Tolle, K. M., Eds. *The Fourth Paradigm: Data-Intensive Scientific Discovery.* Microsoft Research, 2009.

[82] Hjaltason, G. R., and Samet, H. Ranking in spatial databases. In *Proceedings of the 4th International Symposium on Large Spatial Databases (SSD), Portland, ME* (1995), pp. 83–95.

[83] Hua, M., Pei, J., Zhang, W., and Lin, X. Ranking queries on uncertain data: a probabilistic threshold approach. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Vancouver, BC* (2008), pp. 673–686.

[84] Hughey, R., and Krogh, A. Hidden markov models for sequence analysis: extension and analysis of the basic method, 1996.

[85] Iijima, Y., and Ishikawa, Y. Finding probabilistic nearest neighbors for query objects with imprecise locations. In *Proceedings of the 10th International Conference on Mobile Data Management (MDM), Taipei, Taiwan* (2009), IEEE, pp. 52–61.

[86] Imielinski, T., and Jr., W. L. Incomplete information in relational databases. *Journal of the ACM 31*, 4 (1984), 761–791.

[87] Jagadish, H. V., Ooi, B. C., Tan, K.-L., Yu, C., and Zhang, R. iDistance: an adaptive B+-tree based indexing method for nearest neighbor search. *ACM Transactions on Database Systems (TODS) 30*, 2 (2005), 364–397.

[88] Jampani, R., Xu, F., Wu, M., Perez, L. L., Jermaine, C. M., and Haas, P. J. Mcdb: a monte carlo approach to managing uncertain data. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Vancouver, BC* (2008), pp. 687–700.

[89] Jensen, C. S., Lin, D., and Ooi, B. C. Query and update efficient b+-tree based indexing of moving objects. In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB), Toronto, Canada* (2004), pp. 768–779.

[90] Kang, J. M., Mokbel, M. F., Shekhar, S., Xia, T., and Zhang, D. Continuous evaluation of monochromatic and bichromatic reverse nearest neighbors. In *Proceedings of the 23rd International Conference on Data Engineering (ICDE), Istanbul, Turkey* (2007), pp. 806–815.

[91] Karlin, S., and Ost, F. Maximal length of common words among random letter sequences. *The Annals of Probability 16* (1988), 535–563.

[92] Karlin, S., and Taylor, H. M. *A First Course in Stochastic Processes*, vol. 2. Academic Pr Inc, 1975.

[93] Kastenmüller, G., Kriegel, H.-P., and Seidl, T. Similarity search in 3d protein databases. In *Proceedings of the German Conference on Bioinformatics (GCB), Cologne, Germany* (1998), pp. 3–5.

[94] Keogh, E. Exact indexing of dynamic time warping. In *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB), Hong Kong, China* (2002), pp. 406–417.

[95] Korn, F., and Muthukrishnan, S. Influenced sets based on reverse nearest neighbor queries. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Dallas, TX* (2000), pp. 201–212.

[96] Kriegel, H.-P., Kunath, P., Pfeifle, M., and Renz, M. Probabilistic similarity join on uncertain data. In *Proceedings of the 11th International Conference on Database Systems for Advanced Applications (DASFAA), Singapore* (2006), pp. 295–309.

[97] Kriegel, H.-P., Kunath, P., and Renz, M. Probabilistic nearest-neighbor query on uncertain objects. In *Proceedings of the 12th International Conference on Database Systems for Advanced Applications (DASFAA), Bangkok, Thailand* (2007), pp. 337–348.

[98] Kriegel, H.-P., and Seeger, B. Multidimensional order preserving linear hashing with partial expansions. In *International Conference on Database Theory* (1986), pp. 203–220.

[99] Kripke, S. A completeness theorem in modal logic. *Journal of Symbolic Logic 24*, 1 (1959), 1–14.

[100] Krogh, A., Brown, M., Mian, I. S., Sjölander, K., and Haussler, D. Hidden markov models in computational biology: applications to protein modeling. *Journal Of Molecular Biology 235* (1994), 1501–1531.

[101] Kuijpers, B., and Othman, W. Trajectory databases: Data models, uncertainty and complete query languages. *Journal of Computer and System Sciences 76*, 7 (2010), 538–560.

[102] Lakshmanan, L. V. S., Leone, N., Ross, R. B., and Subrahmanian, V. S. Probview: A flexible probabilistic database system. *ACM Transactions on Database Systems (TODS) 22*, 3 (1997), 419–469.

[103] Lazaridis, I., and Mehrotra, S. Progressive approximate aggregate queries with a multi-resolution tree structure. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Santa Barbara, CA* (2001).

[104] Lee, J.-G., Han, J., and Whang, K.-Y. Trajectory clustering: a partition-and-group framework. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Beijing, China* (2007), pp. 593–604.

[105] Lee, K. C. K., Zheng, B., and Lee, W.-C. Ranked reverse nearest neighbor search. *IEEE Transactions on Knowledge and Data Engineering 20*, 7 (2008), 894–910.

[106] Li, J., and Deshpande, A. Consensus answers for queries over probabilistic databases. In *Proceedings of the 28th ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, Providence, RI* (2009), pp. 259–268.

[107] Li, J., and Deshpande, A. Ranking continuous probabilistic datasets. *Proceedings of the VLDB Endowment 3*, 1 (2010), 638–649.

[108] Li, J., Saha, B., and Deshpande, A. A unified approach to ranking in probabilistic databases. *Proceedings of the VLDB Endowment 2*, 1 (2009), 502–513.

[109] Li, Z., and Ge, T. Online windowed subsequence matching over probabilistic sequences. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Scottsdale, AZ* (2012), pp. 277–288.

[110] Lian, X., and Chen, L. Efficient processing of probabilistic reverse nearest neighbor queries over uncertain data. *The VLDB Journal 18*, 3 (2009), 787–808.

[111] Lian, X., and Chen, L. Probabilistic inverse ranking queries over uncertain data. In *Proceedings of the 14th International Conference on Database Systems for Advanced Applications (DASFAA), Brisbane, Australia* (2009), pp. 35–50.

[112] Lian, X., and Chen, L. A generic framework for handling uncertain data with local correlations. *Proceedings of the VLDB Endowment 4*, 1 (2010), 12–21.

[113] Litwin, W. Linear hashing: A new tool for file and table addressing. In *Proceedings of the 6th International Conference on Very Large Data Bases (VLDB), Montreal, Canada* (1980), pp. 212–223.

[114] Ljosa, V., and Singh, A. K. APLA: Indexing arbitrary probability distributions. In *Proceedings of the 23rd International Conference on Data Engineering (ICDE), Istanbul, Turkey* (2007).

[115] Lowe, D. Object recognition from local scale-invariant features. In *International Conference on Computer Vision, Corfu, Greece* (1999), pp. 1150–1157.

[116] MacQueen, J. Some methods for classification and analysis of multivariate observations. In *5th Berkeley Symposium on Mathematics, Statistics, and Probabilistics* (1967), vol. 1, pp. 281–297.

[117] Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C., and Byers, A. H. Big data: The next frontier for innovation, competition, and productivity, 2011.

[118] Moghadam, A., Jebara, T., and Schulzrinne, H. A markov routing algorithm for mobile dtns based on spatio-temporal modeling of human movement data. In *Proceedings of the 14th ACM international conference on Modeling, analysis and simulation of wireless and mobile systems* (2011), pp. 323–332.

[119] Mokbel, M. F., Chow, C.-Y., and Aref, W. G. The new casper: A privacy-aware location-based database server. In *Proceedings of the 23rd International Conference on Data Engineering (ICDE), Istanbul, Turkey* (2007), pp. 1499–1500.

[120] Mokbel, M. F., Ghanem, T. M., and Aref, W. G. Spatio-temporal access methods. *IEEE Data Engineering Bulletin 26*, 2 (2003), 40–49.

[121] Mokhtar, H., and Su, J. Universal trajectory queries for moving object databases. In *Proceedings of the 5th International Conference on Mobile Data Management (MDM), Berkeley, CA* (2004).

[122] Ngai, W. K., Kao, B., Chui, C. K., Cheng, R., Chau, M., and Yip, K. Y. Efficient clustering of uncertain data. In *Proceedings of the 6th IEEE International Conference on Data Mining (ICDM), Hong Kong, China* (2006), pp. 436–445.

[123] Nguyen-Dinh, L.-V., Aref, W. G., and Mokbel, M. F. Spatio-temporal access methods: Part 2 (2003 - 2010). *IEEE Data Engineering Bulletin 33*, 2 (2010), 46–55.

[124] Nieblack, W., Barber, R., Equiz, W., Flickner, M., Glasman, E., Petkovic, D., Yanker, P., Faloutsos, C., and Taubin, G. The qbic project - querying images by content using color, texture, and shape. In *Proceedings of the Storage and Retrieval for Media Databases (SPIE), San Jose, CA* (1993), pp. 173–187.

[125] Nievergelt, J., Hinterberger, H., and Sevcik, K. C. The grid file: An adaptable, symmetric multikey file structure. *ACM Transactions on Database Systems (TODS) 9*, 1 (1984), 38–71.

[126] Papadias, D., Kalnis, P., Zhang, J., and Tao, Y. Efficient olap operations in spatial data warehouses. In *Proceedings of the 7th International Symposium on Spatial and Temporal Databases (SSTD), Redondo Beach, CA* (2001).

[127] Patroumpas, K., Papamichalis, M., and Sellis, T. K. Probabilistic range monitoring of streaming uncertain positions in geosocial networks. In *Proceedings of the 24rd International Conference on Scientific and Statistical Database Management (SSDBM), Crete, Greece* (2012), pp. 20–37.

[128] Pei, J., Jiang, B., Lin, X., and Yuan, Y. Probabilistic skylines on uncertain data. In *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB), Vienna, Austria* (2007), pp. 15–26.

[129] Pelekis, N., Kopanakis, I., Kotsifakos, E. E., Frentzos, E., and Theodoridis, Y. Clustering trajectories of moving objects in an uncertain world. In *Proceedings of the 9th IEEE International Conference on Data Mining (ICDM), Miami, FL* (2009), pp. 417–427.

[130] Pelekis, N., Kopanakis, I., Kotsifakos, E. E., Frentzos, E., and Theodoridis, Y. Clustering trajectories of moving objects in an uncertain world. In *Proceedings of the 9th IEEE International Conference on Data Mining (ICDM), Miami, FL* (2009), pp. 417–427.

[131] Pelekis, N., Kopanakis, I., Marketos, G., Ntoutsi, I., Andrienko, G. L., and Theodoridis, Y. Similarity search in trajectory databases. In *Proc. TIME* (2007), pp. 129–140.

[132] Pfoser, D., and Jensen, C. S. Capturing the uncertainty of moving-object representations. In *Proceedings of the 6th International Symposium on Large Spatial Databases (SSD), Hong-Kong* (1999).

[133] Pfoser, D., Jensen, C. S., and Theodoridis, Y. Novel approaches in query processing for moving object trajectories. In *Proceedings of the 26th International Conference on Very Large Data Bases (VLDB), Cairo, Egypt* (2000), pp. 395–406.

[134] Rahm, E., and Do, H. H. Data cleaning: Problems and current approaches. *IEEE Data Engineering Bulletin 23*, 4 (2000), 3–13.

[135] Ré, C., Letchner, J., Balazinksa, M., and Suciu, D. Event queries on correlated probabilistic streams. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Vancouver, BC* (2008), pp. 715–728.

[136] Roussopoulos, N., Kelley, S., and Vincent, F. Nearest neighbor queries. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), San Jose, CA* (1995), pp. 71–79.

[137] Saltenis, S., Jensen, C. S., Leutenegger, S. T., and Lopez, M. A. Indexing the positions of continuously moving objects. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Dallas, TX* (2000), pp. 331–342.

[138] Samet, H. *Foundations of Multidimensional and Metric Data Structures.* Morgan Kaufmann, San Francisco, 2006.

[139] Sankaranarayanan, J., Samet, H., and Varshney, A. A fast all nearest neighbor algorithm for applications involving large point-clouds. *Computer and Graphics 31*, 2 (2007), 157–174.

[140] Sarma, A. D., Benjelloun, O., Halevy, A. Y., and Widom, J. Working models for uncertain data. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE), Atlanta, GA* (2006).

[141] Seidl, T., and Kriegel, H.-P. Optimal multi-step k-nearest neighbor search. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Seattle, WA* (1998), pp. 154–165.

[142] Sen, P., Deshpande, A., and Getoor, L. Representing tuple and attribute uncertainty in probabilistic databases. In *ICDM Workshops* (2007), pp. 507–512.

[143] Sen, R., and Deshpande, A. Representing and querying correlated tuples in probabilistic databases. In *Proceedings of the 23rd International Conference on Data Engineering (ICDE), Istanbul, Turkey* (2007).

[144] Singh, A., Ferhatosmanoglu, H., and Tosun, A. S. High dimensional reverse nearest neighbor queries. In *Proceedings of the 12th ACM Conference on Information and Knowledge Management (CIKM), New Orleans, LA* (2003), pp. 91–98.

[145] Singh, S., Mayfield, C., Mittal, S., Prabhakar, S., Hambrusch, S. E., and Shah, R. Orion 2.0: native support for uncertain data. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Vancouver, BC* (2008), pp. 1239–1242.

[146] Soliman, M. A., and Ilyas, I. F. Ranking with uncertain scores. In *Proceedings of the 25th International Conference on Data Engineering (ICDE), Shanghai, China* (2009), pp. 317–328.

[147] Soliman, M. A., Ilyas, I. F., and Chang, K. C.-C. Top-k query processing in uncertain databases. In *Proceedings of the 23rd International Conference on Data Engineering (ICDE), Istanbul, Turkey* (2007), pp. 896–905.

[148] Stanoi, I., Agrawal, D., and Abbadi, A. E. Reverse nearest neighbor queries for dynamic databases. In *Proceedings of the ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD), Dallas, TX* (2000), pp. 44–53.

[149] Tao, Y., Cheng, R., Xiao, X., Ngai, W. K., Kao, B., and Prabhakar, S. Indexing multi-dimensional uncertain data with arbitrary probability density functions. In *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB), Trondheim, Norway* (2005), pp. 922–933.

[150] Tao, Y., Faloutsos, C., Papadias, D., and Liu, B. Prediction and indexing of moving objects with unknown motion patterns. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Paris, France* (2004), pp. 611–622.

[151] Tao, Y., and Papadias, D. Time-parameterized queries in spatio-temporal databases. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Madison, WI* (2002), pp. 334–345.

[152] Tao, Y., Papadias, D., and Lian, X. Reverse kNN search in arbitrary dimensionality. In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB), Toronto, Canada* (2004), pp. 744–755.

[153] Tao, Y., Papadias, D., and Shen, Q. Continuous nearest neighbor search. In *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB), Hong Kong, China* (2002), pp. 287–298.

[154] Tao, Y., Papadias, D., and Sun, J. The tpr*-tree: An optimized spatio-temporal access method for predictive queries. In *Proceedings of the 29th International Conference on Very Large Data Bases (VLDB), Berlin, Germany* (2003), pp. 790–801.

[155] Tao, Y., Xiao, X., and Cheng, R. Range search on multidimensional uncertain data. *ACM Transactions on Database Systems (TODS) 32*, 3 (2007), 15.

[156] Tao, Y., Yiu, M. L., and Mamoulis, N. Reverse nearest neighbor search in metric spaces. *IEEE Transactions on Knowledge and Data Engineering 18*, 9 (2006), 1239–1252.

[157] Tiakas, E., Papadopoulos, A., Nanopoulos, A., Manolopoulos, Y., Stojanovic, D., and Djordjevic-Kajan, S. Searching for similar trajectories in spatial networks. *Journal of Systems and Software 82*, 5 (2009), 772–788.

[158] Trajcevski, G., Choudhary, A. N., Wolfson, O., Ye, L., and Li, G. Uncertain range queries for necklaces. In *Proceedings of the 11th International Conference on Mobile Data Management (MDM), Kansas City, MO* (2010), pp. 199–208.

[159] Trajcevski, G., Tamassia, R., Ding, H., Scheuermann, P., and Cruz, I. F. Continuous probabilistic nearest-neighbor queries for uncertain trajectories. In *Proceedings of the 12th International Conference on Extending Database Technology (EDBT), Saint-Petersburg, Russia* (2009), pp. 874–885.

[160] Trajcevski, G., Wolfson, O., Hinrichs, K., and Chamberlain, S. Managing uncertainty in moving objects databases. *ACM Transactions on Database Systems (TODS) 29*, 3 (2004), 463–507.

[161] Trajcevski, G., Wolfson, O., Zhang, F., and Chamberlain, S. The geometry of uncertainty in moving objects databases. In *Proceedings of the 8th International Conference on Extending Database Technology (EDBT), Prague, Czech Republic* (2002), pp. 233–250.

[162] Vlachos, M., Gunopulos, D., and Kollios, G. Discovering similar multidimensional trajectories. In *Proceedings of the 18th International Conference on Data Engineering (ICDE), San Jose, CA* (2002), pp. 673–684.

[163] Weber, R., Schek, H.-J., and Blott, S. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the 24th International Conference on Very Large Data Bases (VLDB), New York City, NY* (1998), pp. 194–205.

[164] Wei, L.-Y., Zheng, Y., and Peng, W.-C. Constructing popular routes from uncertain trajectories. In *Proceedings of the 18th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Beijing, China* (2012), pp. 195–203.

[165] Welch, L. R. Hidden markov models and the baum-welch algorithm. *IEEE Information Theory Society Newsletter 53*, 4 (2003), 1, 10–13.

[166] White, D. A., and Jain, R. Similarity indexing with the SS-tree. In *Proceedings of the 12th International Conference on Data Engineering (ICDE), New Orleans, LA* (1996), pp. 516–523.

[167] Wolfson, O., Sistla, P., Chamberlain, S., and Yesha, Y. Updating and querying databases that track mobile units. *Distributed and Parallel Databases 7*, 3 (1999).

[168] Wu, W., Yang, F., Chan, C.-Y., and Tan, K. L. Continous reverse k-nearest-neighbor monitoring. In *Proceedings of the 9th International Conference on Mobile Data Management (MDM), Beijing, China* (2008), pp. 132–139.

[169] Xia, C., Lu, H., Ooi, B. C., and Hu, J. GORDER: An efficient method for KNN join processing. In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB), Toronto, Canada* (2004), pp. 756–767.

[170] Xia, T., and Zhang, D. Continous reverse nearest neighbor monitoring. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE), Atlanta, GA* (2006), p. 77.

[171] Yang, C., and Lin, K.-I. An index structure for efficient reverse nearest neighbor queries. In *Proceedings of the 17th International Conference on Data Engineering (ICDE), Heidelberg, Germany* (2001), pp. 485–492.

[172] Yeh, M.-Y., Wu, K.-L., Yu, P. S., and Chen, M. PROUD: a probabilistic approach to processing similarity queries over uncertain data streams. In *Proceedings of the 12th International Conference on Extending Database Technology (EDBT), Saint-Petersburg, Russia* (2009), pp. 684–695.

[173] Yu, C., Cui, B., Wang, S., and Su, J. Efficient index-based knn join processing for high-dimensional data. *Information and Software Technology 49*, 4 (2007), 332–344.

[174] Yuan, J., Zheng, Y., Xie, X., and Sun, G. Driving with knowledge from the physical world. In *Proceedings of the 17th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), San Diego, CA* (2011), pp. 316–324.

[175] Zacharias, N., and Zacharias, M. I. The twin astrographic catalog on the hipparcos system. *The Astronomical Journal 118*, 5 (1999), 2503–2510.

[176] Zhang, J., Mamoulis, N., Papadias, D., and Tao, Y. All-nearest-neighbors queries in spatial databases. In *Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM), Santorini Island, Greece* (2004), pp. 297–306.

[177] Zhang, P., Cheng, R., Mamoulis, N., Renz, M., Züfle, A., Tang, Y., and Emrich, T. Voronoi-based nearest neighbor search for multi-dimensional uncertain databases. In *Proceedings of the 29th International Conference on Data Engineering (ICDE), Brisbane, Australia* (2013), p. (to appear).

[178] Zhang, T., Ramakrishnan, R., and Livny, M. BIRCH: An efficient data clustering method for very large databases. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Montreal, Canada* (1996), pp. 103–114.

[179] Zhang, Y., Lin, X., Zhang, W., Wang, J., and Lin, Q. Effectively indexing the uncertain space. *IEEE Transactions on Knowledge and Data Engineering 22*, 9 (2010), 1247–1261.

[180] Zheng, Y., and Zhou, X., Eds. *Computing with Spatial Trajectories.* Springer, 2011.

Acknowledgements

This work would not have been possible without the support and encouragement of many people. I want to express my gratitude to all of them, even if I cannot mention everyone here.

First of all, I want to extend my warmest thanks to my supervisor Prof. Dr. Hans-Peter Kriegel who gave me the opportunity to work on this topic. His way of leading the group and creating an inspiring and productive working environment will always be a model for me.

I am especially thankful to Prof. Dr. Thomas Seidl, who readily agreed to act as a second reviewer.

It is hardly possible to overestimate the significance my colleagues most of whom turned out to be friends rather than co-workers. Their help and support by inspiring discussions as well as humor were essential for the development of this thesis. I am especially grateful to PD Dr. Matthias Schubert and PD Dr. Matthias Renz who worked as a "catalyzers" during the development of new ideas and impressed me more than one time by their out-of-the-box thinking. Retrospectively the collaboration with Andreas Züfle and Thomas Bernecker over the last years was a more than a fortunate coincidence. The complementary background and competences lead to amazingly creative discussions and productive works. Susanne Grienberger deserves a special thanks. Her verve and helpfulness are some of the reasons the past years went by so easily. The importance of Franz Krojer, who kept our equipment running smoothly, is not to be underestimated.

The support by my friends kept me going even during harder times. I really consider myself blessed to be friends with some of the best people on earth. Among many others, Nino Göres, Felix Lusteck and Pia Frei have to be mentioned here. An important counterbalance to the work on this thesis was playing music with "Sound Solution" and I hope we will play on together until the last string broke.

From the bottom of my heart I thank one unnamed person without whom I would be another man. Though we currently tread different paths there will always be a special spot in my heart for you.

Finally and most important, I want to thank my family for their everlasting love and care. I would be nothing without you.

Curriculum Vitae

Tobias Emrich was born on June 9, 1982 in Munich, Germany. He finished his secondary education at the Erasmus-Grasser-Gymnasium in Munich in 2001. In the following year, he fulfilled his alternative civilian service at the Studienseminar Albertinum in Munich.

In 2002, he began studying media informatics at the Ludwig-Maximilians-Universität (LMU) Munich. During his studies, he was working at ifap GmbH in Munich. In 2007, he successfully finished his diploma thesis *Ein Monitoringsystem für Interradiostationen* in the database group of Prof. Dr. Hans-Peter Kriegel.

Afterwards, he started in Prof. Kriegel's group as an assistant lecturer and researcher. During this time, he participated at scientific exchanges with collaborators at the University of Hong Kong. His research comprised efficient pruning strategies for spatial similarity queries, management and querying of uncertain data, uncertain spatio-temporal data and subspace search in high dimensional datasets.