Hierarchical Graphs as Organisational Principle and Spatial Model Applied to Pedestrian Indoor Navigation

Edgar-Philipp Stoffel

München, 2009

Hierarchical Graphs as Organisational Principle and Spatial Model Applied to Pedestrian Indoor Navigation

Edgar-Philipp Stoffel

Dissertation

zur Erlangung des akademischen Grades des
Doktors der Naturwissenschaften
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig–Maximilians–Universität München

vorgelegt von

Edgar-Philipp Stoffel
aus Bukarest

München, den 21. April 2009

Erstgutachter:
Prof. Hans Jürgen Ohlbach (Ludwig–Maximilians–Universität München)

Zweitgutachter:
Prof. Mike Rosner (University of Malta)

Tag der mündlichen Prüfung:
2. Juni 2009

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

In this thesis, hierarchical graphs are investigated from two different angles – as a general modelling principle for (geo)spatial networks and as a practical means to enhance navigation in buildings. The topics addressed are of interest from a multi-disciplinary point of view, ranging from Computer Science in general over Artificial Intelligence and Computational Geometry in particular to other fields such as Geographic Information Science.

Some hierarchical graph models have been previously proposed by the research community, e.g. to cope with the massive size of road networks, or as a conceptual model for human wayfinding. However, there has not yet been a comprehensive, systematic approach for modelling spatial networks with hierarchical graphs. One particular problem is the gap between conceptual models and models which can be readily used in practice. Geospatial data is commonly modelled - if at all - only as a flat graph. Therefore, from a practical point of view, it is important to address the automatic construction of a graph hierarchy based on the predominant data models. The work presented deals with this problem: an automated method for construction is introduced and explained.

A particular contribution of my thesis is the proposition to use hierarchical graphs as the basis for an extensible, flexible architecture for modelling various (geo)spatial networks. The proposed approach complements classical graph models very well in the sense that their expressiveness is extended: various graphs originating from different sources can be integrated into a comprehensive, multi-level model. This more sophisticated kind of architecture allows for extending navigation services beyond the borders of one single spatial network to a collection of heterogeneous networks, thus establishing a meta-navigation service. Another point of discussion is the impact of the hierarchy and distribution on graph algorithms. They have to be adapted to properly operate on multi-level hierarchies.

By investigating indoor navigation problems in particular, the guiding principles are demonstrated for modelling networks at multiple levels of detail. Complex environments like large public buildings are ideally suited to demonstrate the versatile use of hierarchical graphs and thus to highlight the benefits of the hierarchical approach. Starting from a collection of floor plans, I have developed a systematic method for constructing a multi-level graph hierarchy. The nature of indoor environments, especially their inherent diversity, poses an additional challenge: among others, one must deal with complex, irregular, and/or three-dimensional features. The proposed method is also motivated by practical considerations, such as not only finding shortest/fastest paths across rooms and floors, but also by providing descriptions for these paths which are easily understood by people. Beyond this, two novel aspects of using a hierarchy are discussed: one as an informed heuristic exploiting the specific characteristics of indoor environments in order to enhance classical, general-purpose graph search techniques. At the same time, as a convenient by-product of this method, clusters such as sections and wings can be detected. The other reason is to better deal with irregular, complex-shaped regions in a way that instructions can also be provided for these spaces. Previous approaches have not considered this problem.

In summary, the main results of this work are:

- hierarchical graphs are introduced as a general spatial data infrastructure. In particular, this architecture allows us to integrate different spatial networks originating from different sources. A small but useful set of operations is proposed for integrating these networks. In order to work in a hierarchical model, classical graph algorithms are generalised. This finding also has implications on the possible integration of separate navigation services and systems;

- a novel set of core data structures and algorithms have been devised for modelling indoor environments. They cater to the unique characteristics of these environments and can be specifically used to provide enhanced navigation in buildings. Tested on models of several real buildings from our university, some preliminary but promising results were gained from a prototypical implementation and its application on the models.

ZUSAMMENFASSUNG

In dieser Arbeit werden hierarchische Graphen auf zwei verschiedene Gesichtspunkte hin untersucht – zunächst einmal als generelles Prinzip zur Modellierung von (geographisch-)räumlichen Netzen und des weiteren als praktisches Hilfsmittel um Navigation innerhalb von Gebäuden zu verbessern. Die angesprochenen Themen haben multidisziplinäre Aspekte, die von der Informatik im Allgemeinen über die Künstliche Intelligenz und Computergeometrie im Speziellen bis hin zu anderen Gebieten wie die Raumbezogene Informationswissenschaft reichen.

Obwohl einige Modelle von hierarchischen Graphen bereits in der Forschungsgemeinde vorgeschlagen wurden, z.B. um mit der enormen Größe von Straßennetzen zurecht zu kommen, oder als konzeptionelles Modell für die menschliche Navigation, gibt es noch keinen umfassenden systematischen Ansatz um räumliche Netze mittels hierarchischer Graphen zu modellieren. Ein spezielles Problem besteht darin, den Unterschied zwischen konzeptionellen Modellen und praktisch einfach verwendbaren Modellen zu überbrücken. Räumliche Informationen werden – wenn überhaupt – in Form von flachen Graphen modelliert. Demzufolge ist es aus praktischer Sicht bedeutend, den automatischen Aufbau einer Hierarchie von Graphen anzusprechen die auf vorherrschenden Datenmodellen gründet. Die vorliegende Arbeit befasst sich mit diesem Problem: es wird eine automatische Methode zum Aufbau vorgestellt und erklärt.

Ein spezieller Beitrag meiner Arbeit ist der Vorschlag, hierarchische Graphen als Grundgerüst für eine flexible, erweiterbare Architektur zu verwenden um verschiedene (geographisch-)räumliche Netze zu modellieren. Der vorgeschlagene Ansatz ergänzt klassische Graphenmodelle auf elegante Weise, im Sinne daß ihre Ausdruckskraft erweitert wird: verschiedene Graphen die aus verschiedenen Quellen stammen können zu einem umfassenden Modell mit unterschiedlichen Abstraktionsgraden eingegliedert werden. Diese ausgeklügelte Architektur ermöglicht es, Navigationsdienste über die Grenzen eines einzelnen räumlichen Netzes hinweg zu erweitern zu einer Ansammlung von verschiedenartigen Netzen. Somit wird ein Meta-Navigationssystem aufgebaut. Ein weiterer Aspekt ist die Auswirkung der Hierarchie auf Graphenalgorithmen. Diese müssen angepasst werden, um auf einer mehrschichtigen Hierarchie richtig zu funktionieren.

Durch die spezielle Untersuchung von Navigationsproblemen innerhalb von Gebäuden werden die Grundprinzipien zur Modellierung von Netzen auf verschiedenen Abstraktionsebenen erarbeitet. Komplizierte Umgebungen wie große öffentliche Gebäude sind bestens dafür geeignet, die flexible Einsatzweise von hierarchischen Graphen aufzuzeigen, und somit die Stärken des hierarchischen Ansatzes zu unterstreichen. Angefangen mit einer Sammlung von Grundrissen habe ich eine systematische Methode entwickelt, um eine mehrschichtige Graphenhierarchie aufzubauen. Der Charakter von Gebäuden, insbesondere die inhärente Vielfältigkeit, stellt eine weitere Herausforderung dar: man muss sich unter anderem mit komplexen, unregelmäßigen und/oder dreidimensionalen Merkmalen befassen. Die vorgeschlagene Methode ist auch auf praktischen Überlegungen begründet, so z.B. nicht nur die kürzesten/schnellsten Wege über Räume und Geschoße hinweg zu finden, sondern auch Beschreibungen dieser Wege bereitzustellen, die für den Menschen ein verständlich sind. Darüber hinaus werden zwei neue Aspekte bzgl. der Benutzung einer Hierarchie angesprochen: zum einen die Hierarchie als informierte Heuristik, die die besonderen Merkmale vom Inneren von Gebäuden ausnutzt und klassische Allzweck-Suchverfahren auf Graphen verbessert. Dabei werden zusätzlich automatisch Cluster wie verschiedene Bereiche in einem Gebäude oder Gebäudeflügel erkannt. Der andere Grund besteht darin, besser mit unregelmäßigen, komplex gearteten Räumen umzugehen, damit Beschreibungen auch für diese Räume bereitgestellt werden können. Frühere Ansätze haben dieses Problem nicht berücksichtigt.

Die Hauptergebnisse der Arbeit sind folgende:

- zunächst werden hierarchische Graphen als allgemeine räumliche Dateninfrastruktur eingeführt. Insbesondere ermöglicht es uns diese Architektur, verschiedene räumliche Netze aus verschiedenen Quellen zusammenzuführen. Eine kleine nützliche Menge an Operationen wird vorgeschlagen um diese Netze zu integrieren. Um auch in einen hierarchischen Modell zu funktionieren werden klassische Graphenalgorithmen erweitert. Dieser Befund hat auch Folgen für die mögliche Integration von getrennten Navigationssystemen und -diensten.

- danach werden eine neue Reihe von zentralen Datenstrukturen und Algorithmen für das Modellieren von Gebäudeinneren entwickelt. Sie gehen auf die besonderen Merkmale dieser Umgebungen ein und können dazu genutzt werden um die Navigation in Gebäuden zu verbessern. Aufgrund von Tests auf einigen Modellen realer Gebäude unserer Universität wurden erste vielversprechende Ergebnisse mit einem Prototypen und dessen Anwendung auf den Modellen erzielt.

LIST OF PUBLICATIONS

PUBLICATIONS IN CONNECTION WITH THIS THESIS

- Edgar-Philipp Stoffel, Korbinian Schoder, and Hans Jürgen Ohlbach: *Applying Hierarchical Graphs to Pedestrian Indoor Navigation.* In Proceedings of 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM GIS 2008), Irvine, California (5th - 7th November 2008), Organization: ACM, LNCS, ISBN 978-1-60558-323-5 © Springer-Verlag

- Sebastian Mieth, Florian Fuchs, Edgar-Philipp Stoffel, and Diana Weiß: *Reasoning on Geo-Referenced Sensor Data in Physical Infrastructures.* In Proceedings of the Workshop "Semantic Web meets Geospatial Applications" at the 11th AGILE International Conference on Geographic Information Science (AGILE 2008), Girona, Spain (5th - 8th May 2008), Organization: AGILE © Springer-Verlag

- Edgar-Philipp Stoffel and Hans Jürgen Ohlbach: *Versatile Route Descriptions for Pedestrian Guidance in Buildings - Conceptual Model and Systematic Method.* In Proceedings of 11th AGILE International Conference on Geographic Information Science (AGILE 2008), Girona, Spain (5th - 8th May 2008), Organization: AGILE, LNGC, May 2008, ISBN 978-3-540-78945-1 © Springer

- Edgar-Philipp Stoffel, Bernhard Lorenz, and Hans Jürgen Ohlbach: *Towards a Semantic Spatial Model for Pedestrian Indoor Navigation.* In Proceedings of 1st International Workshop on Semantics and Conceptual Issues in Geographical Information Systems (SeCoGIS 2007), Auckland, New Zealand (5th - 9th November 2007), LNCS 4802, 328-337, November 2007, ISBN 978-3-540-76291-1 © Springer

- Bernhard Lorenz, Hans Jürgen Ohlbach, and Edgar-Philipp Stoffel: *A Hybrid Spatial Model for Representing Indoor Environments.* In Proceedings of 6th International Symposium on Web and Wireless Geographical Information Systems (W2GIS 2006), Hong Kong, China (4th - 5th December 2006), LNCS 4802, 328-337, December 2006, ISBN 978-3-540-76291-1 © Springer

- Hans Jürgen Ohlbach, Mike Rosner, Bernhard Lorenz, and Edgar-Philipp Stoffel: *NL Navigation Commands from Indoor WLAN fingerprinting position data.*
  REWERSE Deliverable A1-D7, see `http://rewerse.net/`

other publications

- Christian Hänsel, Hans Jürgen Ohlbach, and Edgar-Philipp Stoffel: *L-DSMS - A Local Data Stream Management System.* Additional Information: `www.pms.ifi.lmu.de/rewerse-wga1/ldsms/publications`. In Software Architecture, Proceedings of the Second European Conference on Software Architecture (ECSA 2008), Paphos, Cyprus (29th September - 1st October 2008), LNCS 5292, 298-305, ISBN 978-3-540-88029-5 © Springer-Verlag

achievements

- **"Best Fast Forward Presentation" Award** at the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM GIS 2008) for presenting the poster "Applying Hierarchical Graphs to Pedestrian Indoor Navigation"

SUPERVISION OF STUDENT WORK

DIPLOMA THESES

- Korbinian Schoder: *Indoor Route Planning for Humans - Conceptual Design and Prototpye*, 2008.

- Sebastian Mieth: *Reasoning on Location-Related Sensor Data in Infrastructure Networks*, supervised together with Florian Fuchs and Diana Weiss, 2008.

- Alexei Poupychev: *Entwicklung eines Frameworks zur Simulation von beweglichen Objekten in Schienennetzen*, supervised together with Florian Fuchs and Diana Weiss, 2008.

- Martin Wassermann: *Konzeption und Realisierung eines Web basierten Mediatorsystems für Graphenalgorithmen über verteilte geographische Netze*, 2007.

- Volker Iden: *A Language and a Framework for Rule-Based Modification of Semantic Models*, 2007.

- Matthias Schmeisser: *PlanML: A Markup Language for Navigational Planning*, supervised together with Bernhard Lorenz, 2006.

- Christian Bode: *An Ontology-based Repository for Web Services*, supervised together with Bernhard Lorenz, 2006.

- Andreas Heindel: *Nutzung von Indoor-Positionierungsdaten zur mobilen Wegeplanung*, supervised together with Bernhard Lorenz, 2006.

PROJECT THESES

- Alexei Poupychev: *Implementing Heuristic Algorithms for the Travelling Salesman Problem*, 2007.

- Boris Ridnyk: *Implementing an Editor for Indoor Navigation Models*, 2007.

- Martin Wassermann: *Extending the TransRoute Graph Visualisation System*, supervised together with Bernhard Lorenz, 2006.

- Volker Iden: *An Ontology of Persons and Vehicles for Movement in Geospatial Networks*, 2006.

- Christian Hänsel: *TMC-KML Verkehrsdatenservice für den Google Earth Client*, supervised together with Bernhard Lorenz, 2006.

Part I

PRELIMINARIES

1 INTRODUCTION

"We shape our buildings, and afterwards our buildings shape us." – Winston Churchill

1.1 MOTIVATION

Navigation systems have become an indispensable part of modern society. They help us find our way from virtually any location A to any other location B. The spatial knowledge contained in these digital maps are especially useful for getting around in foreign or unfamiliar places. In large metropoles for instance, driving can be highly complex given the amount of one-way streets, traffic restrictions, etc.

Value of Spatial Data and Knowledge

Beyond this, the value associated with spatial data and knowledge is immense: Not only are spatial data/knowledge widely used in commercial applications, as in the traditional field of Geographic Information Systems (GISs), but they have also found their way into applications in the private sector. The provision of location-based services, along with their consumption on small electronic devices (such as PDAs and mobile phones) is a prominent example for this development. Apart from this, digital globes and precise models of the real world are now available over the World Wide Web for everybody. One can explore distant, unknown places in these virtual environments in a short time from one's desktop.

Urban Areas and Indoor Environments

In these digital world models, urban areas are among the most complicated environments because of their patchwork character – they are multi-modal, heterogeneous environments which consist of large networks of streets, various systems for public transportation, and last but not least indoor environments. As to be pointed out in the next chapters, the latter environments are the most interesting ones from a research perspective:

Within this thesis, the underlying principles, methods, and data structures are investigated for pedestrian navigation in indoor environments.

Importance of Indoor Navigation

Notably, computer-assisted guidance of people inside buildings has become a new and active field of research. This development has been spurred to a large extent by the technological advances made in recent years – mobile devices are now available which work together with dedicated positioning technologies in buildings (such as W-LAN fingerprinting [ORLS06], RFID [BP00], active badges [Pri05], etc.). All these technologies have been making consistent progress. In addition, there is a growing need for applications which assist humans in wayfinding especially in medium and large scale indoor environments. Quick and reliable guidance for finding one's way in a timely manner is not only needed in transit structures like airports or train stations, but also in museums, hospitals, large office complexes, etc. This includes not only everyday situations and tasks, but also extraordinary situations such as emergencies and evacuation where quick action is particularly required.

Research on Indoor Navigation

Research on indoor navigation is multi-disciplinary and involves several directions: it can be roughly subdivided into research on positioning systems which locate users/devices in buildings [ORLS06, BP00, Pri05], general data modelling and system architecture [CW01, GM03], path finding methods and techniques from Artificial Intelligence [dBvKOS00, CB00], research on human spatial cognition [RW99, WM03a] and wayfinding with landmarks [RW02, EB04], and finally user interfaces and methods for presenting spatial information in an appealing manner [RM04, DGP05]. However, in contrast to the plethora of existing, ready-to-use systems for car navigation, practical systems for guiding pedestrians around cities and buildings are scarce to find. Most of them are still at the stage of development. Although some prototypical systems exist which are e.g. employed in museums as guides, these systems mostly constitute of specialised solutions made for one or a few particular buildings only.

Research Challenges

No universal and comprehensive system for pedestrian guidance and indoor navigation yet exists. This is due to a number of difficulties. First of all, there are substantial differences to navigation in outdoor environments. Notwithstanding their smaller size, indoor environments can be quite complex, especially with regard to the third dimension (multiple floor levels) and fully two-dimensional spaces (such as large halls etc.). Public buildings, such as museums, exhibition halls, airports, hospitals, universities or even 'ordinary' office buildings are often experienced as complex environments. The underlying data models and methods of outdoor navigation cannot be simply adopted. A different representation has to be chosen, taking into account the above mentioned issues that make indoor environments special. Then, there are a great number of different positioning systems, all making use of complementary aspects/infrastructures (for an overview, cf. Ohlbach et al. [ORLS06]). Furthermore, there are no standards, which makes integration a difficult task.

1.2 GOALS AND ADDRESSED QUESTIONS

The research questions addressed in this thesis pertain to the design issues of an indoor navigation system for pedestrians. However, the reader should not expect to find a ready implementation of a complete indoor navigation system at the end of the thesis. In fact the focus is on conceptual issues: modelling the geometry of a floor plan, dealing with the third dimension, and similar problems are discussed here. Since the quality of an assistive system primarily depends on the way knowledge is *internally* represented, the underlying design with data models and algorithms should be well thought out. The general principles underlying such a system design are outlined in this work. Three topics are considered to be of particular importance:

1. **Integration**,
2. **Automation**,
3. **Suitability for Humans**.

INTEGRATION. Navigation systems are, generally speaking, specialised solutions for a particular application domain such as road networks, public transportation, etc. They work rather well in their respective domain, but on the other hand, they are restricted to this domain. There is no concept of a *universal* system which integrates *all* these different kinds of networks into a meta-navigation system. The latter would enable users to plan across the different domains seamlessly.

Following this line of thought, a navigation system for the interior of a building should be designed in such a way that it can be easily integrated with other outdoor networks. The underlying data model should be generic and the system architecture flexible enough to allow for diverse other spatial networks to be integrated into the system.

This flexibility is required for representing a number of more complex environments, including large malls which contain a number of different shops, building complexes on a premise which are connected by an underground system of tunnels or bridges, or hotels as in Las Vegas which also consist of built-in casinos. Moreover, train stations or airports are important hubs, i.e. transit points for changing the modality of transport. They are embedded in different networks.

AUTOMATION. Digital models of a building are for the most part available in form of floor plans, e.g. originating from CAD systems. However, there is an inherent problem with this kind of data representation: while such models concentrate mainly on geometric aspects, the navigation structure is not directly represented in them. Thus, it has to be constructed or derived. Most indoor models use some graph-based representation for this purpose [FMW05]. How can these graphs be derived in general? An ad hoc solution, which is often employed, consists of *manually* overlaying a graph structure on different regions (see Fig. 1).

However, constructing such a model by hand is not only tedious but also error-prone. Besides, what are the criteria for setting

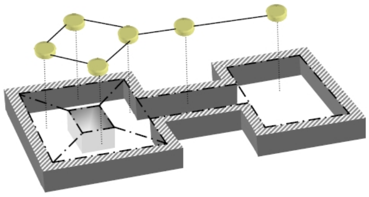


Figure 1: Overlaying a Floor Plan with a Graph [BM05]

nodes and edges? Since there is *no unique* graph representation, i.e. several mappings to a graph are possible, choosing a certain modelling may seem arbitrary. All in all, this situation is unsatisfactory. Instead, a systematic method to derive a graph structure is preferable. It should adhere to clearly defined criteria. Then, floor plans could be interpreted in an automated manner and the navigation structure derived from them.

SUITABILITY FOR HUMANS. For the acceptance of a navigation system, it is crucial that the users' needs are catered for. This is especially important with indoor navigation systems which are designed for pedestrians. First, this concerns issues such as the way instructions are being presented to people (e.g. in form of maps with arrows, or verbal instructions etc.). Thus, path finding alone is insufficient for an indoor navigation system. Paths should be described and represented in a meaningful way so that directions are easy to follow and critical waypoints are passed. The second issue concerns context information, including personal preferences into path finding. Not all paths are feasible for all users. There could be access restrictions, personal preferences, etc. which have to be taken into account. Users should be able to specify their own criteria for path finding.

1.2.1 *Contributions*

Hierarchical Graphs as Core of an Indoor Navigation System

Hierarchical graphs are proposed in this thesis as the basis for an extensible, flexible architecture for modelling various (geo)spatial networks, above all indoor environments. This concerns, from a conceptual point of view, data organisation and maintenance of multiple representations. The different representations can be either distinguished by their granularity (one representation being the abstraction of another one) or by the fact that different types of networks/environments are involved. Hierarchical graphs allow for modelling all these aspects. Therefore they are proposed in this work as the basic data structure at the core of an indoor navigation system. The rest of this thesis introduces hierarchical graphs in a formal way and then discusses the suitability of this model, investigating to what extent a hierarchical graph system fulfills the above mentioned goals of

- integrating different networks into one representation,
- providing methods for an automatic model construction,

- and finally enhancing pedestrian navigation in buildings.

Although all three aspects are explored at appropriate depth, the emphasis in this work lies on the second aspect: It is shown how a hierarchical graph model (comprising multiple levels) can be automatically constructed from a given set of floor plans. Automation has a notable practical impact since it renders unnecessary the tedious work of manually constructing digital navigation models.

1.2.2 *Issues not Covered*

Within the scope of this thesis, not all aspects of indoor navigation could be addressed. Indoor navigation is a very broad and complex field, inherently multi-disciplinary. The focus in this work is therefore on the general (high-level) system architecture, data structures and -models, methods for automation and corresponding algorithms which constitute the core functionality. Complementary aspects on a lower technical level, such as indoor positioning systems, are not further addressed here. Since there are a variety of different positioning systems, choosing the 'right' one is a difficult task. Also, new or enhanced technologies are likely to be developed. The goal pursued here is to abstract from a particular positioning technology as far as possible, so that the general system and functionality remains independent from changing technologies.

1.3 ORGANISATION OF THIS THESIS

This thesis is organised into three parts and eight chapters. Each chapter can be read on its own, since cross-references between different chapters are made explicit where this is necessary for comprehension. Each part builds on the the previous parts. In detail, the structure of this thesis looks as follows:

Chapter 2 presents the basic concepts of pedestrian indoor navigation. Here the particular characteristics of indoor environments are discussed which make navigation a challenging task. Existing models for navigation in buildings are highlighted and compared, revealing their individual strengths and weaknesses. Moreover, the role of context information is discussed, and how it can be incorporated into the models.

Chapter 3 introduces hierarchical graphs by first showing their use in various domains and applications. Then different formalisations of hierarchical graphs are highlighted to give a clearer understanding. Differences and commonalities of different definitions are pointed out. The terminology used in the subsequent part of this thesis is introduced. The first part ends here.

The next part constitutes the main part of the thesis. It starts with Chapter 4, where the general design issues of a hierarchical graph system are addressed. The aspect of integration is covered first and foremost: A mediator-based architecture is presented which allows for integrating different graphs (which can also be modelled at different levels of detail) from different sources into a comprehensive system. The system architecture which has been prototypically implemented is explained here. Then it is illustrated how context information can be integrated in this architecture. A versatile set of operations is provided for incrementally creating a hierarchical graph out of flat graphs. Notably, a distributed architecture is considered where these flat graphs can be physically situated on different machines (sources). To the best of the author's knowledge, this is a novel aspect which has not been considered so far. The details of the operations which can be performed on hierarchical graphs are first formally defined and then discussed with regard to their consistency. The notion of consistency is formalised as well.

Chapter 5 builds on this general architecture and applies hierarchical graphs to the modelling of indoor environments. The addressed issues are twofold: First a systematic method is described which enables one to automatically create a multi-level graph hierarchy from floor plans. It starts from a floor plan's geometry. Different aspects of hierarchisation are discussed here. However, the main thrust is the goal of automation. Then selected examples show how the obtained hierarchical model can be exploited for concrete navigation problems in buildings. This concerns not only path finding, but also the means and concepts which are necessary for the (automatic) generation of human-oriented route descriptions. The descriptions can be obtained by exploiting the hierarchical graph representation. Specific algorithms are suggested for this purpose.

Chapter 6 wraps up the main part by pointing out and reviewing the progress of implementation concerning the different components/-modules. A preliminary evaluation of a hierarchisation method is shown here, and further criteria for evaluation are elaborated.

In the final part of the thesis, related work is presented in Chapter 7. This includes not only topics such as hierarchical graph models (albeit for other spatial networks mainly), but also systems for indoor navigation and human wayfinding in general. Finally, Chapter 8 concludes the thesis. The main results are summarised and directions for future research are pointed out.

2 BACKGROUND

This chapter presents the basic concepts of pedestrian indoor navigation, along with a general notion of context information for wayfinding. The special characteristics of indoor environments (in contrast to outdoor spatial networks) are highlighted first, together with the implied requirements for a pedestrian navigation system. Then a number of existing graph-based models for navigation in buildings are compared, revealing their complementary strengths and weaknesses. The suitability of these models is investigated especially with regard to *pedestrian* indoor navigation. This involves not only means for finding shortest/fastest etc. paths, but also for describing these paths to human wayfinders. Moreover, different ways of representing context information are discussed. It is pointed out, in general, how these criteria can be reflected in so-called cost functions, that is optimisation goals for path finding in graphs.

2.1 CHALLENGES OF PEDESTRIAN INDOOR NAVIGATION

2.1.1 *Motivation: Characteristics of Indoor Environments*

Car navigation systems are an indispensable aid for travelling nowadays. They have reached the stage of maturity and have become a mass product: The navigation problems encountered in this domain have been analysed in considerable detail. Consequently, plenty of solutions are available for delivering turn-by-turn instructions to drivers. Due to the simplicity and efficiency of these techniques, they are employed on a large scale, for example in commercial car navigation systems or traffic information systems.

Turn-by-turn Instructions from Spatial Networks

The key to the success of these systems lies in the underlying data model: Graphs are a convenient model for many environments which people travel through. In a geographical context, the basic mapping of spatial networks to a graph structure is often straightforward. Take for example the street network of a city: nodes simply represent intersections and edges represent road segments. Given that the spatial orientation of an edge in the network determines the course of motion along the edge, it is a fairly simple task to provide turn-by-turn instructions. It would be indeed desirable to apply this technique to other domains like indoor environments, expecting that it would work equally well there.

Particularities of Indoor Environments

Unfortunately, it turns out that navigation problems in large buildings are conceptually quite different from those problems encountered in street networks (see e.g. [GM03]). The crux of the matter lies in the topological structure of indoor environments; it is much more diverse than the topological structure of road or railway networks. The following points illustrate the aspects which distinguish indoor environments from ordinary spatial networks:

DIVERSITY VS. UNIFORMITY OF SHAPE. Roads and tracks have standard properties like width, number of lanes, et cetera.[1] The network structure is clearly defined and regular: every junction is a decision point, and road segments are linear. Large metropoles often show a mesh or grid pattern. Although architects of buildings adhere to general principles, they are given more freedom in designing a building. Depending on the intended function of rooms, they can vary considerably in size, in their amount of connections to other rooms or in their shape. All sorts of different room shapes can occur in floor plans. Particularly large rooms can be unique with regard to their irregular shape and their multitude of connections. Because of this, a systematic treatment of indoor environments is difficult.

FREE VS. CONFINED MOVEMENT. Especially in large halls and comparable sorts of *open space*, pedestrians can move *freely*. There is no guarantee that they will stick to a preconceived path network laid over a floor plan. This makes the design of a suitable navigation system more challenging. Pedestrian motion is less restricted than, say, the motion of vehicles within a network (for instance cars or tramways). Not only are vehicles mostly bound to lanes/rails, but drivers also have to obey traffic regulations, e.g. speed limits or turn restrictions. Driving a car, one is not allowed to turn just anywhere; manœuvres such as reversing the direction are often not permitted or practically not even possible. Note that restrictions, although pertaining to *access*, can also be found in buildings. The typical division into public and private areas is a prime example. Fig. 2 contrasts a large room in an indoor environment to a junction in a road network.

GRANULARITY. Pedestrians move at a much slower pace than, for example, automobiles. Consequently their perspective of the environment is richer in details. This in turn means that the features of a building have to be modelled at a finer granularity (for example, comprising the niches of a room or places where items can be stored).

HIERARCHY VS. NETWORK. Features like roads and railways can be boiled down to *one*-dimensional path structures. This is very convenient for providing turn-by-turn instructions. On the other hand, it is much more difficult to extract path structures from buildings because they exhibit genuine *two*- and *three*-dimensional structures: Except for corridors, stairs, or doors,

[1] however, in different countries, there can of course be different norms, such as for the width of rail tracks

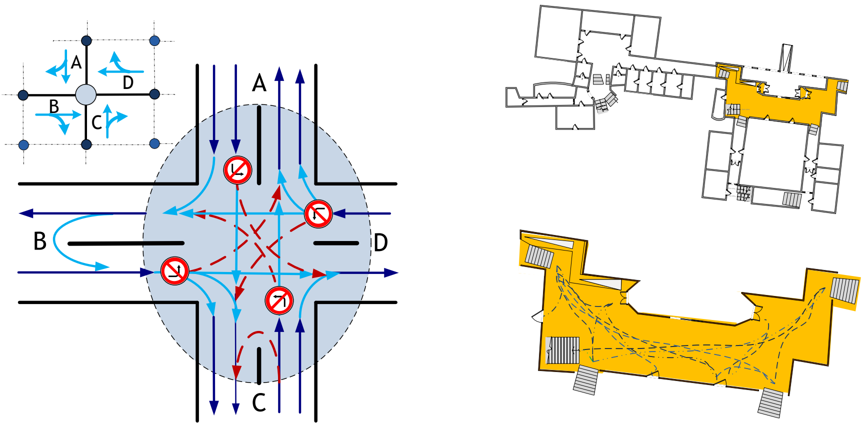


Figure 2: Motion in Road Networks vs. Indoor Environments

> no evident navigational structures are present in buildings. The dominant theme is rather a hierarchy, with layers of nested regions.

Open vs. Closed Spaces

One factor responsible for the particular difficulty of navigation in buildings is the coexistence of two different types of spaces – *open* spaces and *closed* spaces[2]: Timpf and Rüetschi [RT04] have pointed to this distinction in their analysis of Swiss railway stations. While they argue that network space (i.e. closed space) is mediated by maps, e.g. to represent train connections between cities, they also point to the fact that open space (which is called scene space) is composed of individual regions. Train stations are a good example due to the typically large dimensions of halls and lobbies.[3] A vast number of different ways are conceivable for crossing these open spaces. In contrast, corridors, stairs, and other linear features fall into the first category of closed spaces. They represent a network structure with an implicit direction to move in. Therefore they are less difficult to deal with. However, it is problematic to draw a quantitative distinction line between the two types of spaces. Because the transition is based on human perception of space, it seems rather fuzzy.

Coexistence of Various Graph Representations

Unsurprisingly, there several correct ways for overlaying a floor plan with a graph. Franz et al. [FMW05], for example, present a series of different representations from a multidisciplinary standpoint. It is arguable what should be represented by a node or an edge, respectively: e.g. consider the concrete possibilities of mapping a corridor to a node, to a sequence of connected nodes, or rather to an (hyper)edge. Things get even more complex when we consider, for example, halls with a rather irregular shape or three-dimensional features across floors. Which representation should be chosen? A simple answer to these questions cannot be given. Rather, it should be decided ad hoc which representation fits best: every model has its

[2] not to be understood in the sense of *public* vs. *private* space, but rather referring to the degree of freedom for pedestrians moving in these spaces

[3] comparable examples from outdoors are navigation in deserts with no roads or steering a boat across a lake.

individual advantages and drawbacks. We shall go into more detail in the following and compare different models proposed in literature from the perspectives of path finding and giving route directions.

2.1.2 *General Modelling Principles*

The preceding points have shown that modelling indoor environments is not trivial. Indoor navigation is indeed a large research field which involves many different aspects. Among these, there are for example diverse positioning technologies ranging from W-LAN fingerprinting [ORLS06] to touch-sensitive areas. Quite a wide range of alternative solutions have emerged in recent years. We do not cover these technical aspects further in this thesis, but concentrate instead on the general modelling principles because the latter constitute a solid, common foundation for *any* system to be implemented (also in the future, given that technology evolves constantly and the perfect system is yet to come). For a more detailed list and discussion of positioning technologies, cf. Ohlbach et al. [ORLS06].

Before we begin with a thorough evaluation of different models of indoor environments, we must first define the fundamental properties of a graph representation. The main criteria considered in particular are threefold:

1. the basic representation of the *environment* (2D and/or 3D).

2. the facilitation of *path finding* in the environment.

3. the means to derive meaningful *route descriptions*.

ENVIRONMENT. First and foremost, the basic representation comprehends a mapping of different architectural features to nodes or edges in a graph structure. Notice that nodes and edges can carry completely different *semantics* in different models (e.g. a door could be represented either by an edge or by a node, depending on whether general access between rooms is considered or rather waypoints along a path). It is important to make clear the semantics in the respective model. A representation could merely reflect the environment, but it could additionally comprise a more or less sophisticated model of human wayfinding behaviour, e.g. represent certain locations inside a region which are visually or otherwise salient.

Modelling Aspects

Granularity is another important aspect to consider, since it determines which features are actually relevant for the model and which ones can be omitted. For a more pragmatic point of view, there are further aspects of interest like the *modelling effort* (expressed in time and space complexity), the possibility of *automation* (i.e. how much manual intervention is necessary during the modelling process), and finally the *expressiveness* of the model (i.e. what sort of knowledge can be derived from it). It is all these things that make up the characteristics of a model.

PATH FINDING. Concerning the suitability of a model for path finding, we have to reconsider the role of geometrical distance vs. topological distance. The accurate modelling of distance is crucial in domains such as robot navigation (e.g. for driving safely past or along a wall), since small deviations can result in undesired consequences. However, different criteria play a role for human indoor navigation: determining the *absolute* shortest path (by precision of centimetres) is not of the utmost importance. A path which is *almost* as short as the optimal one can still qualify as 'good enough', as long as no alternative path is *substantially* faster.[4] An abstract path that approximates a geometric path (within a certain range) might be sufficient.

Let us assume that the geometrical shortest path passes through an office or lecture hall. In this case, it may not be appropriate at all to choose this path. Life in a society imposes additional restrictions on path finding.

Restrictions

Among others, restrictions due to physical, legal, social, and temporal conditions must be considered [Rau01, HEH03]. In this particular example, the doors to the office could be closed, or going through the lecture hall would disturb ongoing courses or examinations. It would be sensible to have a concept reflecting the separation of public as opposed to private rooms. In all these cases an alternative path through a corridor would be preferred, even though it is *not* the geometrically shortest path. These examples suggest that for a realistic application, many different factors have to be taken into account. It would be desirable to have a generic, flexible approach which allows for specifying more sophisticated and realistic cost criteria beyond a simple optimisation goal as distance. In this respect, a symbolic approach for path finding could complement a purely numeric optimisation. Section 2.2.5 discusses this topic in more detail.

ROUTE DESCRIPTIONS. The generation of human-oriented route descriptions for guidance in public buildings is still an open, active field of research. Compared to path finding, this problem is much more sophisticated; it involves a handful of techniques from various disciplines, such as natural language generation [DGP03, Miz04], qualitative spatial representations [SMR06] and reasoning [ZF96, CH01] and many more.

Meaningful Labels and Qualitative Spatial Relations

Route descriptions pose additional requirements to the model: First, a set of different attributes and labels needs to be provided which can then be turned into their linguistic counterpart. Formal knowledge representation systems, such as ontologies, could define a vocabulary for the key concepts which are referred to in expressions. Second, reasoning needs to be conducted pertaining to the spatial relations of rooms and paths through 2D/3D space. Not all of these properties can be hard

[4] a qualitative measure of distance, such as that proposed in Hernández et al. [HCF95] can help to define the appropriate meaning of *substantially* shorter / faster in an application.

coded in the model, since this would take up a combinatorial amount of space. The fact that indoor environments are much less confined than other domains aggravates this problem. Thus, it is vital to build the model in a clever way that facilitates the derivation of essential knowledge for route descriptions.

The styles and forms of route descriptions are quite endless – there is room for plenty of variety. The so-called process of spatial chunking plays a prominent role in this respect, since it defines the essential criteria for the contraction of route descriptions. Relevant aspects for chunking in indoor environments could be, besides the wayfinder's level of familiarity with the environment, the arrangement of spaces into higher-order concepts like sections, wings, floors, et cetera.

Various studies have investigated the form of textual/verbal instructions for indoor environments [MSK06]. Scarce evidence can be found in literature [Miz04, SMR06, MSK06] for a systematic method of generating human-oriented route descriptions in indoor environments. Most systems rely on a simple graphical interface, where the environment or parts of it are depicted in a map-like manner. However, the conciseness of verbal route descriptions and their dimensionless nature speak in favour of them. Note that they can also be used in combination with traditional graphical interfaces, enhancing this form of representation.

2.1.3 *Evaluating and Adapting Existing Approaches*

Multi-disciplinary Viewpoint

With the general modelling principles made clear, we can now examine for the domain of indoor environments the suitability of existing approaches, i.e. how far they meet the postulated criteria from the previous section. Remarkably, most of the approaches originate from the field of Artificial Intelligence, especially from robot motion planning. There are of course also models from the domain of architecture, although their focus lies mainly on the aspects of design and construction rather than on wayfinding. A notable exception is the work of Passini [Pas84, Pas96], where wayfinding is accredited a central role. In recent years, valuable contributions have also been made in the field of cognitive science and psychology, particularly on the way people organise space and solve problems in space.

In our analysis, we concentrate predominantly on the approaches for robot motion planning, because of their strong algorithmic background. However, we incorporate as well some key ideas flavoured by the other domains into our reflections.

Roadmap

For motion planning applications, so called *roadmaps* [Lat90] are the most commonly used solution for finding paths through 2D space. A roadmap is an one-dimensional discretisation of a higher-dimensional environment into a geometrically anchored graph structure. This way, a two-dimensional structure of regions can be mapped to different locations or representative places within the regions. Two basic classes of roadmaps can be distinguished, namely visibility graphs and Voronoi diagrams:

VISIBILITY GRAPHS [Nil69], first proposed by Nilsson in 1969, established as a standard method for robot path planning [dBvKOS00]. The underlying idea is simple: nodes represent the convex corners of a region, including the corners of all obstacles, and additionally all portals/entries to the region. Two nodes are considered as mutually visible if their direct connection does not intersect any of the region's boundaries. Thus, an edge joins a pair of nodes which can see each other. Fig. 3 depicts an exemplary visibility graph for an indoor environment:

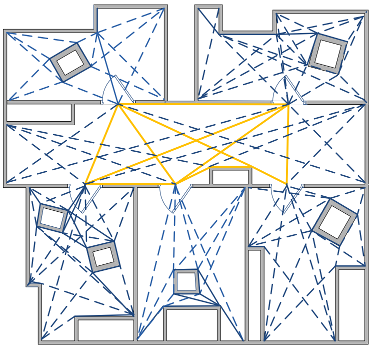


Figure 3: Visibility Graph

Visibility graphs facilitate the precise steering of a robot, because distance and angles are modelled accurately (provided that data quality is sufficient). The obtained geometric paths are optimal in terms of distance. However, the style of the directions for guiding people along a path is quite different from the style of the instructions for controlling a mobile robot: Humans cannot follow instructions like 'the next door is *at an angle of* 134° and a *distance of* 9.2 *metres*' (measurements could be imprecise, anyway).

Proposed Adaptation

Apparently, the spatial relations utilised in the instructions first need to be translated into a qualitative representation. For this purpose, the outgoing edges of a node could be e.g. stored in a list sorted according to increasing angles and distance to the node. The circular order determined by this representation can be used for an enhanced communication of the spatial relations: this is beneficial for the production of statements like 'take the *second* door to your right'.[5] In order to distinguish between the left hand- and right hand side, we additionally need a reference orientation at the respective node. For portals, it could be the direction perpendicular to the respective wall since this is the expected course of movement through the portal (when

[5] for paths in indoor environments, it generally makes more sense to refer to relative, *egocentric* directions ('front', 'back', 'left', 'right') rather than to absolute cardinal directions (North, South, West, East).

entering a room). This method is particularly well-suited for convex spaces with no obstacles.

Inherent Limitations

Local navigation rules, such as the circumvention of obstacles, are basically present in the model, but it is difficult to turn them into an appropriate description referring to the obstacles ('pass between the two obstacles'). The main problem is that points, and not higher-level objects (such as obstacles) are related to each other. In order to solve this problem, nodes need to be annotated as belonging to the same obstacle, or being of a common type, e.g. 'door'. Non-convex corners only play a role for locations which are not mutually visible (otherwise, they do not appear in paths).

A major disadvantage of visibility graphs is the amount of space required for storing nodes and edges: Primarily in convex spaces with no obstacles, the number of possible combinations is quadratic (a complete graph K_n with n nodes has $\frac{n(n-1)}{2}$ edges). All these edges are explicitly modelled, which is an overhead.

A possible remedy is to remove all direct connections between portals, so that only paths of length 2 and above need to be stored. However, caution is required for the paths going through the corners of an obstacle: if these paths were removed information for moving around the respective obstacle would be lost. For all pairs of non-connected portal nodes, mutually visibility would be implicitly assumed. This would result in something what could be called 'invisibility graph' [BHHKM04]. Note that different sets of mutually visible nodes can emerge from this contraction. Since these sets may well overlap, their representation is, in general, subtle [AAAS94].

GENERALISED VORONOI DIAGRAMS [ÓSY87] are a logical extension of traditional Voronoi diagrams beyond point sets. They have been devised for guiding robots safely along a characteristic path, called medial axis, of a 2D space. The medial axis is, more formally, understood as the set of points equidistant to the two closest boundaries[6] (cf. the left hand part of Fig. 4). As such, the medial axis can contain both straight and parabolic edges; the construction is rather sophisticated and consequently takes considerable time. It has been be shown [CB00] that the medial axes meet at common nodes and are arranged in a tree-like structure.

Certainly not all branches of this tree are equally important: For path planning, e.g. it is convenient to cut off all branches that lead to convex corners [Wal04, Whi06]. The result of this simplification step is depicted on the right hand part of Fig. 4: The remaining nodes are intersections of different medial axes (these can be, for example, the two opposite sides of portals).

[6] imagine the inflation of a large balloon in a room until it touches two walls: the centre of the balloon is located right on the medial axis.

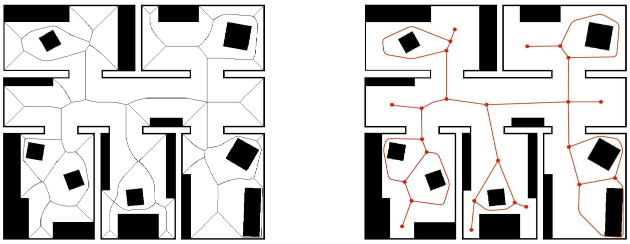


Figure 4: Simplifying a Generalized Voronoi Diagram [Wal04]

As opposed to visibility graphs, Generalised Voronoi Diagrams are quite well-suited for route descriptions. This is especially the case for closed spaces: Consider, for example, the shape around two obstacles. Valid paths are represented concisely.

However, open spaces are clearly the weak spot of Generalised Voronoi Diagrams. This sort of situation is indicated in Fig. 5, on the bottom left and right. Consider the left hand path marked by dashed lines:

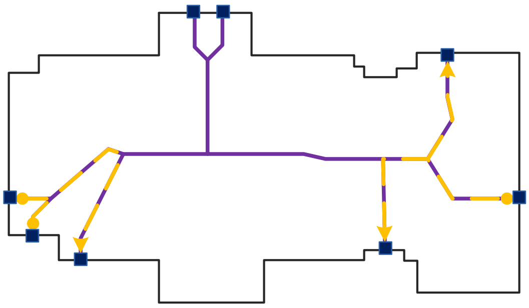


Figure 5: Unnatural Route Descriptions with Generalised Voronoi Diagrams [Whi06]

Illustrating the Difficulties for Open Spaces

Before the destination is actually reached, the path goes into the opposite direction. This zig-zagged course seems quite erratic: While a direct connection of near-by portals would be expected, paths are distorted towards the middle of the open space. Hence the medial axis does not represent a typical path taken by a person. It is evident that these geometrical paths are not suitable for direct translation into route directions. Imagine the (automatically generated) output of a guidance system based on this kind of representation: this would lead to an awkward description like “go left, then turn right” (the question is just where to turn, because of the lack of landmarks), where something similar to “take the second door to your right” would be expected instead. The particular cause for this phenomenon is the bias of paths on the medial axis towards the centre of the open space.

A possible way to deal with this problem is to introduce an additional step of path relaxation [Whi06]: intermediate waypoints of a path are removed consecutively, as long as the shortened path stays within the boundaries. This way, zig-zagged sections of a path are simplified as far as possible.

A more abstract representation for the spatial configuration of rooms is often used in architecture, with so called *access graphs*. They can be conveniently extracted from planar drawings such as floor plans: they are simply the dual planar graph of the drawing. A dual planar graph represents regions as nodes (i.e. whole rooms, corridors) and connections between regions as edges (e.g. portals).

Unlike their geometrical counterpart, access graphs are qualitative, i.e. they merely model the topological relations between different regions. In this kind of representation, distance is rarely considered. It cannot be defined in a context-sensitive manner (e.g. to distinguish whether we go from room A to room B through portal p_1 or p_2), but only as a reasonable approximation such as the Euclidean distance between the centroids of spaces. If the inner structure of regions is intricate, this approach has limitations.

Duality

However, access graphs are complementary to the other geometric approaches, i.e. they can be combined. This would yield a two-level hierarchy of graphs, where several nodes of the finer-grained graph (which belong to the same region) are comprised as a single node in the dual planar graph. This sort of combination is particularly appealing, since it allows instantly addressing different subgraphs that form one region. Apart from doors, the labels and types of regions are also frequently used in route descriptions.

2.1.4 *Summary*

The preceding evaluation revealed that it is, to some degree, possible to adapt existing roadmap methods from robot indoor navigation. However, some intermediate steps are necessary for the transition from robot to human indoor navigation and there are further difficulties to deal with. The core problem is that a model for pedestrian navigation serves a different purpose than a model suitable for steering a robot: ideally, it should encode qualitative spatial relations so that meaningful descriptions can be derived from this model. Visibility graphs and generalised Voronoi diagrams, two common models for robot navigation, have been investigated for their suitability as a pedestrian navigation model. Each of the presented methods has its individual strengths and weaknesses for specific kinds of spaces. The main results of the analysis are summarised in Table 1:

None of the examined approaches performs well in all situations where route descriptions need to be generated. The main problem is the limited flexibility of the models. If possible, it would be desirable to put together the complementary strengths of the approaches into a single, comprehensive model.

Model	**Semantics**		**Type**	**Construction Time**	Suitability for **Path Finding** and **Descriptions**
	Nodes	Edges			
visibility graphs	convex corners, portals	mutual visibility	geometric	$\mathcal{O}(n^2)$ [Wel85], $\mathcal{O}(n \log n)$ for sparse graphs [GM91]	⊖ complete graph, $\mathcal{O}(n^2)$ space ⊖ no common (sub)paths ⊕ accurate distance
Generalised Voronoi Diagrams	portals, where axes meet	medial axes	geometric	$\mathcal{O}(n^2 \log n$ $a)$ [ÓSY87] (a is the inverse Ackermann's function)	⊕ more concise, $\mathcal{O}(n)$ space ⊕ spatial relations derivable (to some degree) ⊖ too much clearance in open spaces
dual planar graphs	regions	portals	topologic	directly extractable from floor plans	⊕ most compact ⊕ no superimposed network, just reflects environment ⊖ distance, if any, approximated ⊖ coarse, i.e. no relation between portals

Table 1: Comparing Different Graph Representations

2.2 CONTEXT INFORMATION AND COST FUNCTIONS

This section discusses the role of context information for human wayfinding. More specifically, examples are illustrated for indoor environments. Possible representations of context information are explained and, more importantly, it is shown how context information can be involved in the process of finding a certain path in a graph.

2.2.1 *The Context of Wayfinding*

Distinguishing Different Meanings of the Word 'Context'

The word 'context', being inherently vague, leaves room for a variety of different interpretations in different domains (e.g. in Human Computer Interaction, context stands for device properties, users' skills and knowledge, etc.) – it thus needs clarification. However, giving a precise definition of 'context' is a difficult task.

In literature various definitions of context can be found (see e.g. [Dey01, Dou04]). These definitions are, for the most part, very general: context is regarded as something relative, defined in terms of a specific situation or task. In the scope of this thesis, we try to narrow down the meaning of context information. We define context strictly in terms of wayfinding:

> *With context information, we mean any piece of information from and about the surrounding environment (including the wayfinder) which is relevant for wayfinding tasks in this environment. Such factors are above all the characteristics and preferences of human wayfinders, as well as the physical, social, temporal and operational conditions of the current environment. Basically, context information comprehends the set of conditions which can have an influence on wayfinding decisions in a concrete situation. The set is by no means exhaustive because it is virtually impossible to enumerate all conceivable criteria of influence. The degree up to which this information is encoded depends on the specific requirements of an application.*

Still, context can mean a variety of different things such as local traffic conditions, occupancy of rooms, departure times fixed in train schedules, etc. There is a great deal of flexibility for capturing the knowledge of a particular application domain. On the other hand, this working definition ensures that context comprehends every property which can be formally represented and processed.

2.2.2 *Examples for Indoor Environments*

What 'Context' Means for Indoor Environments

In the specific case of indoor environments, we can give some typical examples for context information and analyse what should/could be taken into account for path finding. Consider the following list of examples:

- weather conditions (snowfall, storm opposed to sunshine etc.) could determine a **preference** for partial paths either staying within roofed parts of a building or going through an interior court/garden (even if this entails a detour). Likewise, one normally prefers to take the elevator instead of stairs to reach the upper floors of a skyscraper,

- more importantly, a handful of **social restrictions** have to be kept in mind by human wayfinders: one may not enter, for instance, a library with a coat and backpack but must lock them away first. It is likely that one does *not* choose a path going through the office of some employees, nor through a lecture hall, laboratory, patient's room, operating room, etc. (even though these could be shortcuts for a path finding algorithm from a *purely geometrical* point of view),

- comprising **access restrictions** can also be a significant requirement for a navigation model: entering certain parts of a building may require a key, security check or other form of identification and/or authorisation. For example, laboratories and certain facilities within airports are only accessible for a defined group of people. Context is understood in this sense as the *role* of the wayfinder when we mind the distinction between public spaces and private spaces ('*staff only*'),

- the individual **situation of the wayfinder** has to be taken into account too: rooms on different floors have to be reachable via ramps and elevators for persons in a wheelchair. When travelling with small children and/or heavy luggage, it is much easier to take an elevator instead of stairs, although they may be nearer,
- from a practical viewpoint there are, of course, **temporal restrictions** to keep in mind, such as closing times for shops in a mall, parts of a building which are open only on certain weekdays, hours, or seasons,
- moreover, there are **physical restrictions**. As such, the transportation of bulky objects to a location inside the building (due to dimensions) should to be well planned. They may not fit into an elevator, or a staircase may be too narrow for moving the object around the corner.

The above list provides a pragmatic scheme for the classification of context information into a few broad categories. Most of these pertain to some form of **common-sense knowledge**, which can be traditionally encoded in formal ontologies (see next paragraph). We do not claim that incorporating *all* these exemplary conditions would yield a 'perfect' navigation system for a building:

First, context is non-exhaustive. A domain model is inevitably a drastic simplification of reality. This means that it is always possible to find some special case or detail which has not yet been modelled so far. There are also inherently vague or subtle notions, such as certain weak preferences which are very difficult to express formally in an optimisation criterion (e.g. more complicated cases for the first example). Of course, the more factors are taken into account, the more realistic the final result may be.

Moreover, context is highly subjective. *One* particular application or query alone may not require all features to be taken into account, but only a subset.

So the gist of this reflection is to provide a navigation system which offers a flexible framework for specifying and incorporating context information into its path finding modules, rather than trying to model every possible aspect in a hard-wired manner. The framework should be ideally designed in a way that extensions with further context information can be made when required.

2.2.3 *Using Formal Ontologies*

Modelling Context Information with Formal Ontologies

In the field of ubiquitous computing, remarkable efforts towards a comprehensive user model have been made, e.g. in the user modelling ontology called GUMO [HSB$^+$05, Hec05]. Formalisations like these are a first step towards dealing with context information. They can be used for the subsequent processing of context information in path finding and -selection. Note that in general, the creation of formal ontologies for a particular application domain is a complicated undertaking; it usually requires expert knowledge and a considerable amount of time and effort. However, once it has been completed, a number of applications can ideally profit from the common knowledge base created.

In the following, we summarise different approaches for path finding which include a more sophisticated treatment of context information. The intention is to relate knowledge representing context in a generic way with the behaviour of path finding components.

2.2.4 *Multi-criteria Path Finding*

Cost Functions

Path finding is, in the simplest case, performed over a graph $G = (N, E)$ with a single, total **cost function** $c : E \mapsto \mathbb{R}$ to optimise. The cost function c assigns each edge $e \in E$ a numeric value $v \in \mathbb{R}$ indicating the cost or weight associated with the respective edge. In the spatial domain, this measure of cost is usually the geometric distance covered. This classical version of the shortest path problem has been profoundly studied for several decades, with a lot of efficient algorithms proposed for this purpose. The most commonly applied solutions are Dijkstra's famous shortest path algorithm [Dij59] for non-negative costs and the Bellman-Ford algorithm [Bel58, JF62] for the general case where costs can be negative as well. Improvements in the running time could be gained in many practical cases by an intelligent use of advanced data structures, such as Fibonacci heaps [FT87].

An excellent introduction and overview of different families of generalised shortest path problems is given in Pallottino and Scutellà [PS97]: as it is argued, realistic applications often require more than a simple cost function – they need to consider not only one, but a couple of different criteria simultaneously [MP08] which may interplay. This class of problems is generally called **multi-criteria** shortest path problems [Dia79, Lou83]. The traditional algorithms therefore need to be adapted as far as this is possible. However, there are already some cases where a single criterion cannot be specified with a simple cost function such as defined above:

CONTEXT-SENSITIVE cost functions are more expressive: they additionally allow one to assign costs for traversing a *node*, for example a crossing in a street network. These functions are called context-sensitive because, depending on the choice of incoming and outgoing edge, the costs can vary (think for example of turn restrictions or penalties for left turns where the particular choice does make a difference). They are in general of the form $c : E \times E \mapsto \mathbb{R}$. Note that they do not cover the complete set $E \times E$, but are partial functions defined only for incident edges (of a common node). Thus, a cost matrix with pairs of incident edges would have to be stored in every node. A major disadvantage of this solution is that existing algorithms cannot be applied ad hoc – they need to be adapted.

Modelling via Dual Graph

An alternative solution would be to construct the dual graph, which naturally expresses relations between edges in the primal graph as edges in the corresponding dual representation.[7] The GIS community has recently debated this approach for navigation in road networks which requires more elaborate cost functions, such as minimum turn angles [Win01, DK03]. It is more appealing, since traditional algorithms can be applied to the dual graph without any need for modification.

The context encoded in this kind of cost functions is mainly spatial, although it reflects some legal restrictions pertaining to traffic regulations. More generally, a single edge could have different *modalities*, for example representing the suitable means of transport, or the group of persons possessing special access privileges (such as a key, an ID card, or something similar). These aspects can be represented as attributes of the corresponding node or edge in the graph. A cost function can evaluate these attributes and map them to a numeric value (for example, a cost function would yield a value of ∞ if access were not permitted). Time-dependent properties (e.g. schedules of trains) can be difficult to evaluate [PS97], for example if waiting times are considered and different connections have different speeds (say, there are express trains which can overtake others). In these cases, optimisation can be quite challenging. Some particular problems in this connection are very complex and still call for further investigation. They are not discussed in more detail in the present work.

Dealing with Multiple Criteria

We now return to the remaining question of how to deal with multi-criteria problems. This issue is complex, too. The general optimisation problem involving multiple criteria was first studied by Vincke [Vin74]. It has been applied to the shortest path problem by Dial [Dia79] and Hansen [Han79]. Beyond this, it has received considerable attention in various other fields including game theory, economics, and decision theory [Vin92].

In the multi-criteria case, a cost function $c : E \mapsto \mathbb{R}^n$ does not yield a scalar value. Instead, the result becomes a cost vector consisting of $n \geqslant 2$ individual cost components for each criterion:

$$v_{\vec{mult}} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix}.$$

An inherent problem emerges with this extension: it is not always possible to compare composite costs which differ in different components (e.g. two cost vectors like $\binom{3}{4}$ and $\binom{5}{2}$). Note that there is no *total* ordering among these cost vectors, as opposed to the scalar case. Rather, two *non-dominated* or *Pareto-Optimal* solutions exist.

Dominance relation

A path p_a is said to *dominate* another path p_b if, for the corresponding cost vectors v_a and v_b, the following property holds:

[7] Edges in the original graph correspond to nodes in the dual graph. Nodes in the original graph correspond to the set of all their pairs of incident edges. Each pair in the set is an edge in the dual graph.

$\forall i \in 1 \ldots n : v_{a_i} \geqslant v_{b_i}$.

Dominance induces only a *partial* order among paths. Intuitively, a non-dominated path cannot be further improved in one cost component i without degenerating at least some other component j. The resulting family of solutions reflects a weaker sort of optimality – it forms an equilibrium of so-called *Pareto-optimal* solutions. In the absence of a total ordering relation, the number of alternative solutions is quite large: As shown for the case of two criteria [Han79], the number of efficient, non-dominated paths grows exponentially with the number of nodes $|N|$ in the graph (the complexity is $\mathcal{O}(2^{|N|})$). Therefore, already the bicriterion shortest path problem lies in NP. There are several ways for coping with this problem:

NP-completeness of multi-criteria path finding

AGGREGATION INTO A COMPOSITE COST FUNCTION. The simplest conceivable solution is to aggregate all criteria into *one* global cost function (i.e. reducing the vector of all costs to a scalar value). This can be done, for example, by forming a linear combination of all criteria in terms of a weighted sum: $v_{mult} = \sum_{i=0}^{n} \alpha_i v_i$. Within the sum, the coefficients $\alpha_i \in [0 \ldots 1]$ determine the relative importance of the respective criterion i (e.g., if all α_i were 1, then all criteria would be equally important). Other sorts of conversions are also possible, e.g. a product would be employed instead of a sum if asked for the probabilities of failure or a certain quality of service. While this method allows one to deal with multi-criteria problems in the same way as single-criteria problems, it suffers from serious drawbacks:

- this method needs profound knowledge of the domain, or a considerable amount of experimentation to determine the weight factors α_i accordingly.
- it produces one single solution, contrary to the nature of the problem: the aggregated meta-criterion tends to be somewhat artificial. Therefore, it may not be acceptable for decision makers, or even counter-intuitive when conflicting criteria are considered.

Notice that 'acceptable' is an ultimately subjective notion. In place of a single compromise solution, several alternatives should therefore be offered from which expert decision makers carefully assess the appropriateness for the given problem.

DETERMINING THE PARETO-OPTIMAL SOLUTIONS. Another approach is to accept the complex nature of the problem and to determine the family of Pareto-optimal solutions in a hopefully efficient manner. For this purpose, extensions of the A^* heuristic to multi-criteria search [SCCW91, MdlC05] have been proposed. Furthermore, genetic algorithms [FF95, MW06] have been devised and used to compute Pareto-optimal solutions for practical instances of multi-criteria problems. An important idea in this respect is to reduce the cardinality of the solution set by means of clustering algorithms. The clusters represent selected parts of the solution space and hence make the choice considerably easier for the decision maker. Multi-criteria shortest path problems can also be reformulated and solved by means of soft constraint logic programming, as recently shown in Bistarelli et al. [BMR02].

RANKING OF SOLUTIONS VIA PREFERENCE. If more information is available about the involved cost criteria, problem solving can be facilitated substantially: This is the case, for example, when **user preferences** come into play [AW00]: the notion of preference among different criteria can be used for focussing search. That is, if one cost criterion, deemed less important than another one, is violated it may still be tolerable to accept this solution. Such an approach is proposed, e.g. in Alechina et al. [AL01], where the ranking between different cost criteria is defined as a partial order relation. This relation is then combined with the dominance relation.

2.2.5 *Summary*

As has been pointed out, context information can be encoded by means of cost functions in a variety of different ways. Admittedly, choosing the 'right' kind of modelling for an actual application domain is a very complicated task – it requires profound knowledge of the domain. It is up to a couple of true domain experts to check the applicability and rightness of a certain model, or else user studies have to be conducted in order to evaluate and give feedback on the chosen criteria for complex cost functions. Studies for different wayfinding criteria have been conducted among others by Golledge [Gol95b, Gol95a] and Hochmair [HR02, Hoc04].

It is consequently wiser to leave the task of specifying correct cost criteria to domain experts. Our philosophy is a different one: since cost criteria can be very subjective, in the sense of preferences which do not commonly apply to *all* persons, but only to a certain group or individual, there should also be a means to specify them in a flexible way as different cost functions. The goal is to provide a pragmatic framework for specifying and applying different cost functions for different users. They could, for example, exclude certain paths for certain users only, while more general cost functions apply to other users.

Although the expressiveness of this language is important, one should not neglect another issue which is equally important: specifying cost functions should be a difficult task (and ideally supported by graphical tools, too) given that domain experts are often not or only insufficiently familiar with information technology. Section 4.1.3 presents, among others, a solution based on SWRL rules [Ide07] and the notion of treating cost functions as proper objects which can act as parameters for path finding algorithms, following the strategy design pattern [GHJV95]. The advantage gained by using SWRL is that rules can be comfortably edited a graphical editor, thus keeping the technological burden on domain experts rather low.

3 FUNDAMENTALS OF HIERARCHICAL GRAPHS

This chapter presents *hierarchical graphs* – the main data structure which this thesis concentrates on. The versatile use of hierarchical graphs in various domains and applications is highlighted. Then a set of formal definitions is provided to clarify the different notions and meanings of hierarchical graphs. To acquire a general understanding, various notions of hierarchical graphs as given in literature are compared, trying to distill common properties. Along with the various definitions of hierarchical graphs, an introduction of the most relevant terminology used in the subsequent part of this thesis is provided.

The chapter concludes with the implications of using hierarchical graphs for the spatial domain as a basis of a comprehensive navigation system. Implications are discussed from two complementary points of view, i.e. an interior and an exterior perspective: the interior perspective concerns an appropriate system design (with algorithms and data structures/organisation in mind). The exterior perspective is adopted by users, who pose location- and network-related queries to the system (asking, e.g., for the shortest path).

3.1 MOTIVATION: FROM GRAPHS TO HIERARCHICAL GRAPHS

Graphs are Ubiquitous

Graphs are a well-established and universal formalism for representing a plethora of different environments, systems, components, structures and many more. They can be used for nearly every application to model objects and relationships among these objects. It is indeed very easy to find examples of graph models:

- Geographic Information Systems (GISs) are general-purpose systems that cover a broad spectrum of spatially referenced objects. Among others, they provide graph-based models for spatial networks like transportation infrastructures [MS01].

- in traditional computer science, compilers use parse trees as intermediate representation for ultimately generating machine instructions from a piece of code written in higher-level programming languages.

These are just two arbitrary examples picked from the (virtually endless) list of applications which retrieve, process or make use of graph-structured data in any form. There are many other examples;

these include not only models of real world networks like social networks, phone call representations, biochemical pathways, proteins and molecules, but also more generic forms of knowledge representation, e.g. mind maps, semantic networks, or conceptual graphs.[1] All these examples essentially use graph-based representations for their purposes.

Rationale: Beyond Graphs

However, in some cases the expressiveness provided by graphs turns out to be insufficient. Complex applications have very specific requirements which cannot be met by standard graph representations. The latter need to be enhanced into more elaborate forms of representation in order to meet these strict requirements. To give an example, one major reason is that graphs, with growing size and complexity, at some point become impractical to handle. Dealing with massive amounts of information cannot be just done in an ad hoc manner. A comprehensive evaluation, for instance of a complex biochemical pathway like the citric acid cycle in the human metabolic system, would be more difficult to conduct without any prior clustering or grouping of data. Likewise, the need to shift from graphs to hierarchical graphs arises for systems which organise large collections of heterogeneous data.

Enhanced Representation

In all these respects, *hierarchical* graphs offer an enhanced means for modelling very large and complex structures at several levels of granularity. What, in a nutshell, is to be understood as a hierarchical graph? Hierarchical graphs can be intuitively thought of as an extension of graph representations. They can, for example, be obtained by imposing a hierarchical structure on a graph, e.g. a particular decomposition of the graph into several subgraphs [Ste99]. The resulting data structure is more expressive since it allows individually addressing every part or subgraph within the graph. In case the partitioning into subgraphs is recursive, i.e. subgraphs can encompass other subgraphs on their own, it follows that the resulting hierarchy consists of multiple abstraction levels. Supplementing this informal introduction, the following section accounts for a formal definition of hierarchical graphs and discusses further issues pertaining to these. However, this brief explanation should be sufficient for the moment to get a rough idea of what hierarchical graphs are and understand the argument.

Motivating Examples

In order to gain a more comprehensive insight into the range and applicability of hierarchical graphs, consider the following examples. They illustrate specific situations which call for an enhanced representation offered by hierarchical graphs:

COMPUTER NETWORKS AND SOCIAL NETWORKS. The Internet, due to its enormous size and importance for today's life, is probably one of the prime examples for a graph structure. Its structure basically consists of numerous Web pages (represented as nodes) and edges called hyperlinks for navigating between different

[1] cf. `http://conceptualgraphs.org/`

pages. Figure 6 gives an impression of the overall composition and topology of the Internet:

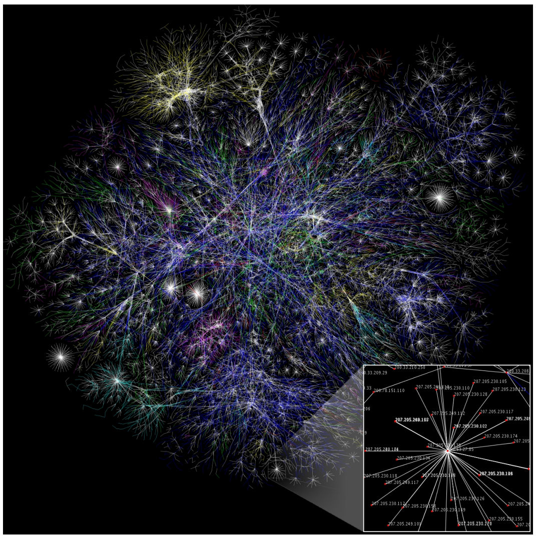


Figure 6: Illustration of the Internet topology (from Wikipedia)

Without breaking down the entire network into different domains and sub-domains, it would be quite difficult to get a clearer picture of its topology, let alone to estimate its dimensions while it is constantly expanding.

But not only the analysis of the network would be hampered. Imagine a network of these dimensions without a hierarchical organisation: sending data packets from one computer to another one would not be feasible, because routing algorithms simply would not scale to meet these dimensions. In other words, communication via the Web would not be possible the way it is done today.

Hierarchical Network Routing

Notably, hierarchy has been a key issue in the design of the Internet. One example instantly springs to mind: Among others, the Domain Name System (DNS) provides a human-friendly, hierarchical naming scheme for locating and addressing computers which participate in the Internet. Humans can memorise a grouping of different resources and/or machines more easily under a common domain name, compared to a handful of plain, differently-looking IP addresses.

The decentralised nature of the Internet becomes especially apparent when we consider routing between different domains (a.k.a. inter-domain routing): Rather than being a single system, the Internet consists of a patchwork of Autonomous Systems[2] (ASes). In order to be able to send and receive data packets across systems' borders, hierarchical routing protocols like the Border Gateway Protocol (BGP) have been devised. They explicitly take advantage of the hierarchical network organisation for routing [KK77]: the next machines (hops) on the way to the

[2] cf. for example `http://www.caida.org/analysis/topology/as_core_network/` for a recent analysis of the higher-level topology

destination machine are chosen according to their connecting path in the hierarchy (this can be extracted from the prefix or postfix of the name).

Social Networks

With the advent of the so-called Web 2.0, the social aspect of the Internet has gained increasing importance. Likewise, we have graphs for modelling individual people and their relationships to other people. Network analysis in this field is predominantly preoccupied with drawing conclusions, such as detecting shared connections (which could stand for friendship or business contact) or people who span bridges across different communities, and other interesting patterns [BCP03]. Clusters play an important role in this respect. The same repertoire of network analysis techniques also applies more or less to telephone call graphs as well as other forms of social networks.

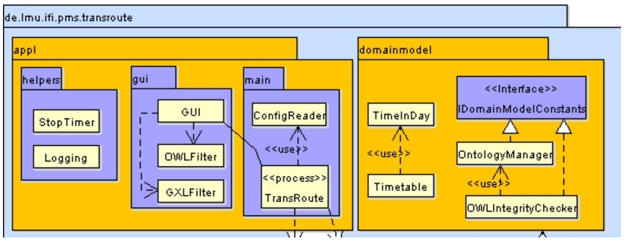


Figure 7: Nested Packages in UML Class Diagrams

Diagrams for System Design

Software engineering. Graph-based formalisms can be found throughout the history of software engineering for various aspects of specifying complex systems: For example, relational data can be conveniently described by means of entity-relationship (ER) diagrams. In the object-oriented programming paradigm, the dependencies, inheritance relations, and associations between different classes are graphically represented in UML class diagrams [RJB04].[3] The latter are used to model, accompanied by UML state charts, the composition and current state of a system more accurately. Note that nested states are allowed in state charts and also packages in UML class diagrams can be nested (see Fig. 7). To complement the modelling of static aspects of a system, Petri nets and activity diagrams, among others, are used for better describing the dynamic behaviour and interaction of components.

Separation of Concerns

Even though not all of the systems are necessarily large and thus difficult to handle, hierarchical graphs still have the advantage that single components, modules and programs can be represented individually. The separation of a system into smaller, meaningful components (by applying the principle of divide and conquer) constitutes a core principle of successful large scale software design. Hierarchy is a very useful concept for expressing and, more importantly, organising this additional information in a convenient manner.

[3] **U**nified **M**odelling **L**anguage

Information Hiding

One of the greatest achievements in this area are modules and components. They allow for encapsulating programs or, more generally, put forth the principle of information hiding: different views on the same information are provided, e.g. based on access rights (think of the common access modifiers `public`, `protected`, and `private` in contemporary programming languages like Java). Depending on the type of a component, it may only be accessed via its interfaces without further insight (*black box* component), or else the detailed interior structure may be revealed to exterior modules (*white box* component). This distinction is important, among others in software testing where both kinds of components need to be treated in their own right and checked somewhat differently.

VISUALISATION. Human-friendly navigation and exploration of large-scale, complex data is becoming more and more important, especially with the previous examples in mind. The visual way of conveying information to people can play a major role in getting them to understand the essential meaning therein. Therefore, a well organised graphical layout can help people considerably to interact even with large amounts of data.

Researchers in fields such as Human Computer Interaction have devoted their efforts to a better understanding of these issues [Nas01]. Visualisation has also been a key issue in the graph drawing community [Ead96] and has gained considerable attention beyond.

Exploration via Folding and Unfolding

Hierarchical graphs are ideally suited in this respect [EF96, YWR03, Rai04]: They allow for visually abstracting large networks by collapsing subgraphs to a single node and, vice versa, expanding a complex node to reveal its interior graph structure if needed. This technique is particularly appealing because it resembles the way humans inherently organise and deal with information [HJ85] by making *abstractions*.[4] Folding and unfolding makes dynamic navigation possible within the graph, at multiple levels of detail.

Practical Concerns

In the above examples it is more appropriate to utilise hierarchical graphs instead of ordinary graphs. Hierarchical graphs are also especially useful under practical considerations: For large networks, despite the mostly 'good' polynomial complexity of basic network algorithms (e.g. shortest path computation), the performance of these algorithms may still be unfeasible in practice. The response time for simple queries may be unacceptable, or even worse, the entire network may be too large to fit into the system's memory. To tackle this problem, it is necessary to improve the way the data is organised. A hierarchical representation is preferable under these circumstances: it

[4] While computers are quite useful for storing and representing massive amounts of information, people face difficulties when it comes to understanding and memorising all this information. However, they adopt *abstraction* as a key strategy to cope with these massive amounts of information: only those pieces of information need to be memorised which are really relevant for a given task (or, at least, those which provide a rough estimation of the overall structure). Everything else can be omitted.

allows breaking down one large graph into smaller, more manageable pieces that fit into the working memory.

Intuitive Representation

Even if the size of the graphs is comparatively small, it may still be beneficial to use a hierarchical representation. If subgraphs represent meaningful abstractions akin to human conceptualisations, they can facilitate navigation tasks for people. Also, if similar graph queries are posed repeatedly, the hierarchy could help to select only the relevant parts affected by the query. The hierarchy could, hence, act as an index structure for obtaining a speed-up on these types of queries.

3.2 BASIC DEFINITIONS AND TERMINOLOGY

3.2.1 *Preliminary Considerations*

What is a hierarchical graph?

Since there are different opinions on this matter, it is difficult to give a precise answer: on the one hand, hierarchical graphs are intuitive to understand informally (see Fig. 8). However, their formalisation is rather intricate because they can be defined in various ways. Despite a remarkable number of approaches in literature to formalise hierarchical graph structures [PL94, FCE95, SM95, JHR98, Ste99], there is no consensus on a generally accepted definition. When considering a suitable formalisation for hierarchical graphs, one has to be aware of certain small subtleties implied by the definitions.

First Impression

Before dealing with the intricacies of the mathematical definitions in detail, let's first try to get an impression of what hierarchical graphs typically look like. For this purpose, consider the prototypical example of a hierarchical graph depicted in Fig. 8:

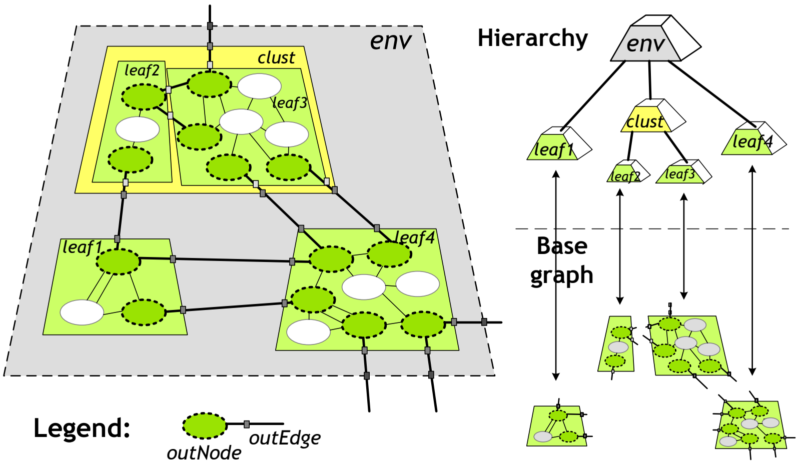


Figure 8: Exemplary Hierarchical Graph

We can see that the (sub)graphs $\mathtt{leaf}_1$ to $\mathtt{leaf}_4$ together already bear the content of the entire graph env. However, there is still an additional clustering (clust) of $\mathtt{leaf}_2$ and $\mathtt{leaf}_3$. Apart from this, the hierarchy is quite simple. Some nodes in the subgraphs have also connections to nodes in other subgraphs; they are referred to as *outNodes* in the legend of Fig. 8, and the respective edges between two such nodes are called *outEdges*.

Two Different Levels

Note that the same information on the hierarchical graph can be visualised and thus conveyed in (at least) two different ways – on the left hand side as an inclusion diagram in the plane (where the subgraphs are represented as areas enclosing their respective nodes and edges), and on the right hand side as a three-dimensional, pyramidal tree structure (where the subgraphs, too, are represented by nodes in the tree and inclusion between different subgraphs is represented by tree edges). There are two distinct levels. We should therefore be careful not to confuse the nodes and edges *in the tree*, which are on the meta-level of the hierarchy and essentially describe subgraphs and their interrelations, with ordinary nodes and edges *in the subgraphs* themselves.

3.2.2 *Three Approaches to Hierarchical Graphs*

As previously suggested, various definitions of hierarchical graphs can be found in literature, however all with a slightly different scope and a different view. In order to gain a common understanding of hierarchical graphs, we now need to systematically analyse commonalities and differences among these definitions. In the following, the main characteristics are illustrated that underlie the most commonly found definitions:

GRAPH MORPHISM/TRANSFORMATION. Hierarchical graphs are, generally speaking, a series of different graphs at *different levels of detail* or *granularity*. Two graphs of subsequent levels l and $(l+1)$ are linked together in the sense that the nodes at the higher abstraction level $(l+1)$ correspond to subgraphs at the lower abstraction level l. In other words, the two graphs (let's call them G_l and G_{l+1}) can be obtained from each other by either refining certain nodes in G_{l+1} to subgraphs of G_l, or, the other way around, by abstracting subgraphs at level l to nodes at level $(l+1)$. Apparently, one can define mappings or rules that explicitly state how such a transformation process is carried out:

Definition 3.2.1 (Hierarchical Graphs via Graph Morphism/-Transformation). Given a graph $G_l = (N_l, E_l)$, we can obtain the hierarchical graph G_{l+1} of the next coarser level $(l+1)$ by applying a series of so-called *graph morphism* functions m_l. These functions map subgraphs $S_l = (N_s \subset N_l, E_s \subset E_l)$ of G_l to nodes of G_{l+1}:
$m_l : s_l \in S_l \longmapsto n_{l+1} \in N_{l+1}\}$. The graph morphism functions m_l are required to be *bijective*, which ensures a direct one-to-one correspondency between subgraphs s_l and nodes n_{l+1}.

Note that the definition only tells us how to obtain *nodes* of the next level graph G_{l+1}, but what about the edges of G_{l+1}?

Implicit Edges

By choosing certain subgraphs of G_l to be coarsened, the graph morphism m_l already implies (although it is not made explicit) how to treat edges: we can distinguish between *internal* edges of a subgraph and *external* edges (the *outEdges* in Fig. 8) which cross the borders of the subgraph. The latter kind of edges are, for this reason, also referred to as boundary-crossing edges. With this additional information in mind, there is a natural way of defining edges for the higher-level graph G_{l+1} – for every *internal* edge e_l within a subgraph s_l (connecting two nodes in s_l), there is *no* corresponding edge in G_{l+1}.

However, all external edges leading out of the subgraph s_l to another subgraph r_l are represented as edges in G_{l+1} between the nodes $m_l(s_l)$ and $m_l(r_l)$ (which are the images of the subgraphs s_l, resp. r_l in m_l).

Overlapping Subgraphs?

In the most general form of definition, different subgraphs of G_l may principally **overlap**. However, in many cases, definitions are more restricted for practical reasons: either the overlapping part of subgraphs $S_{li} \cap S_{lj}$ may only contain nodes, but *no* edges [JHR98], or overlapping is prohibited completely [Rai04]. Even if there are no overlaps between subgraphs, one does not lose expressiveness: it is possible to represent any overlapping $\bigcap_n S_{ln}$ of a number of subgraphs S_{ln} in G_l *explicitly*, by an additional subgraph (the intersecting nodes and edges are removed from the involved subgraphs S_{ln}, then). There can be $2^{|subgraphs|}$ combinations, though.[5] Anyway, all definitions require that the partitioning into subgraphs is **complete**, i.e. covers the entire graph G_l (or else information on the most detailed graph would be lost in the higher levels of coarsening).

Construction of a Hierarchical Graph

Summarising, the function m_l can be used to obtain a coarsening of a graph G_{l+1}, whereas its inverse counterpart m_l^{-1} describes the backward mapping which can be interpreted as refinement operation of a higher-level node. Together, the two functions are expressive enough to fully cover the relationships among graphs of different abstraction levels. There is a direct consequence: This viewpoint allows us to construct hierarchical graphs in a bottom-up manner, beginning with a detailed *base graph* G_0 (which is initially flat). The construction can be performed by recursively applying different coarsening functions m_l to some subgraphs of G_l (an example for a multiple application of coarsening would be $m_n(m_{n-1}(\ldots(m_0)\ldots)))$. Once the hierarchy is defined, it can be navigated *top-down*, from the coarsest representation, by refining nodes to reveal their matching subgraphs in the next lower level. For applications which allow for visually navigating through a hierarchical graph by means of folding and unfolding, this kind of definition is an ideal starting point.

[5] This is the reason for explicitly representing overlaps as separate subgraphs. Otherwise, all overlaps have to be computed during processing.

NODE PARTITIONING. Alternatively, a hierarchy of graphs can be defined via a (recursive) *partitioning*, e.g. over the node set of a base graph. Partitioning also yields different subgraphs. This technique is, for example, convenient when we have certain clusters of nodes or at least a scheme (according to some specific criteria) for obtaining such a clustering. All knowledge pertaining to the hierarchical organisation is held separately from the ordinary graphs. It is encoded in the partitioning itself:

Definition 3.2.2 (Node Partitions, Induced Subgraphs). Given a base graph $G = (N, E)$, a hierarchy is defined by a (recursive) partitioning $\mathcal{P} = P_1, P_2, \dots, P_k$ of the graph's node set into k different node partitions $P_i \subset N$. Each of these partitions *induces* a subgraph S_i. Especially, this subgraph comprehends all nodes of P_i and all *internal* edges of P_i, that is edges connecting pairs of nodes which both lie in P_i: $S_i = (P_i \subset N, E_i \subset E) | E_i = \{(n_s, n_t) | n_s, n_t \in P_i\}$.

Pros and Cons

This view of a graph hierarchy is pretty similar to the one suggested on the left hand side of Fig. 8 – subgraphs are equivalent to node partitions. Each subgraph/partition represents a selected portion of the base graph. Partitions and partitions of partitions form the levels of the hierarchy. The root is the coarsest subgraph which is not part of any other subgraph. The level of a partition is intuitively understood as its depth w.r.t. the root.

Note that partitioning is understood in a strict mathematical sense here (jointly exhaustive, pairwise disjoint partitions). More precisely, the partitions cover all nodes, and there is no overlap at nodes for different partitions (otherwise, we would not obtain a strictly tree-shaped hierarchy). Note that not all edges are covered by the partitions: there are some edges between nodes in different partitions. They do not belong to any partition, but are only modelled in the base graph.[6] As we shall see, these special nodes of a partition P_i which possess also edges to other partitions are of interest for path finding. They are called **border nodes**: $N_b(P_i) = \{n_b \in P_i | \exists e = (n_b, n_j) \in E \wedge n_j \in P_j \neq P_i\}$.

The strength of the previous definition of node partitions lies in the explicit distinction between hierarchy and graphs: while the hierarchy is described completely by the partitioning into subgraphs and does not resort to any form of nodes or edges for representation, the graphs alone can contain nodes and edges. However, on the downside, it is not clear where to place the edges that connect different partitions other than in the base graph. Particularly in the case of multi-level, recursive partitionings, situations are conceivable where an edge simultaneously

[6] There are also alternative definitions of partitions covering all edges. However, they result in overlapping partitions which we want to avoid.

connects *different* partitions of different abstraction levels. Although everything is encoded in the base graph, it may not be feasible to represent the entire base graph (e.g. due to its size or other concerns of a practical nature) only for these edges. A separate encoding of the subgraphs would be preferable in this case, but where do we then put the edges which span across different partitionings?

EQUIVALENCE CLASSES. All nodes in one partition are considered equivalent under some clustering criterion. Hence, an alternative formalisation of the partitions is also possible by means of equivalence relations:[7]

Definition 3.2.3 (Node Equivalence). We assume that a base graph $G = (N, E)$ is given, together with k binary **equivalence relations** $\sim^1, \ldots, \sim^k \subset (N \times N)$. Two nodes $n_s, n_t \in N$ belong to the same class iff they are k-equivalent, i.e. $n_s \sim^k n_t$ (infix notation).

From this definition, it is easy to obtain the aforementioned partitions: Assigning the set of *internal edges* E_i of a subgraph s_i to the relation $\sim^i$ makes the equivalence class coincide with the corresponding node partition P_i that is associated with s_i. Note that we do not only need *one* equivalence relation, but indeed k relations to describe all of the k different node partitions accordingly. So this is just another way for defining hierarchical graphs, but in essence the expressiveness is the same as for partitions and graph morphisms.

Equivalence classes prove their usefulness when constructing a hierarchy: they can serve as a similarity criterion for the clustering of previously unknown or unrelated data. On the other side, the explicit statement of equivalence between nodes might lead to a vast number of relations, which is not the optimal choice from a storage point of view.

In short, all three presented formalisations – albeit different – are equally expressive in the end [Ste99]. As Fig. 9 suggests, it is fairly easy to move from each formalisation to the other ones:

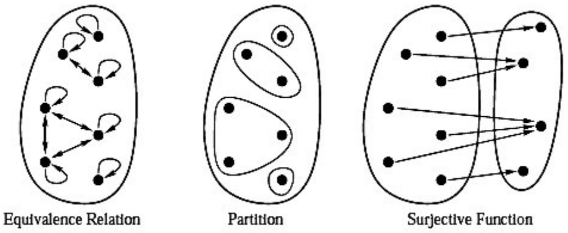


Figure 9: Three Approaches for Defining Hierarchical Graphs [Ste99]

However, choosing the 'right' style of formalisation which suits one particular application best is highly dependent on the characteristics

[7] by definition, an equivalence relation is reflexive, symmetric, and transitive.

of the application and its specific requirements. The bottom line is that *subgraphs* can be regarded as a primitive to the *construction* of a graph hierarchy, irrespective of the way these subgraphs are actually obtained (be it from partitioning, equivalence relations, or graph morphism). Common to all these definitions is the fact that the hierarchy can be constructed from a base graph in a *top-down* manner.

3.2.3 *Further Definitions and their Classification*

There is yet another handful of definitions for hierarchical graphs which are similar, but in other ways different from those previously given. All rely on a common basic formalisation using nodes to describe both subgraphs *and* nodes appearing in subgraphs. Besides this different approach to formalisation, the main difference lies in the way a hierarchy can be constructed according to these definitions. So let's have a closer look at these definitions: First, there is a particular concern regarding the nomenclature of hierarchical graphs. It is mainly due to the coexistence of multiple names, all suggesting similar notions of a hierarchy:

- *nested graphs* [PL94],
- *compound graphs* [SM95],
- *clustered graphs* [FCE95].

The last two types of hierarchical graphs, **compound-** and **clustered graphs**, have a common formalisation as a basis which departs only at one point. The common formalisation looks like this:

Definition 3.2.4 (Hierarchy as Tree). Given a graph $G = (N, E_G)$, we can additionally define a hierarchy as a tree $T = (N, E_T)$ spanning over the same set of nodes N. This essentially means that some nodes in G are subgraphs which may contain other nodes in G. Notice that the relation E_T of tree edges represents inclusion, so it must be *directed*.

Alternatively, the entire information can also be represented in just one graph if we properly distinguish between the two kinds of edge relations: E_G stands for node adjacency and E_T for hierarchic inclusion (these are two *semantically* different notions). This results in a particularly succinct notation for hierarchical graphs, where everything is encoded in one graph.

Albeit succinct, this notation is rather irritating because it mixes two conceptually different levels and forces them into one sole representation: on the one hand, we have the meta-level of the hierarchy (the tree T), with subgraphs and relations among subgraphs. On the other hand, there are the ordinary nodes which are leaves in the hierarchy (i.e. they have no more descendants in T). Both are represented as nodes in N (the same node set is shared by both G and T). As this example suggests, a strict separation between the hierarchical structuring itself and the individual graphs is preferable.

This representation is, notably, more expressive than the previous ones. The underlying reason is that adjacency edges (E_G) can be defined in a variety of different ways: more precisely, for every pair of nodes in N, irrespective of a node in N being a subgraph or a proper node, we can define an edge (also across different hierarchy levels!). However, this representation has an inherent problem: there is no uniform way to model edges. Determining the actual meaning of edges can thus cause trouble. In particular, the question arises whether a higher-level edge between two subgraphs should be explicitly defined if it can be derived from existing lower-level edges which connect nodes in the two subgraphs. The answer depends on the precise type of hierarchical graph.

Clustered graphs and compound graphs both share in principle the common definition from above. They differ, however, in the restrictions imposed on edges:

- **Clustered graphs** are a special subclass of compound graphs. They allow edges (E_G) only between *leaves* of T, i.e. nodes which do not have further subgraphs.
- In contrast, **compound graphs** are more general, offering more freedom for defining edges: any two nodes in G can be connected by an edge $e \in E_G$ as long as they are unrelated in the hierarchy (i.e. no descendants or ancestors in T).

Intricacies of Coarse Edges

Consequently, clustered graphs are semantically equivalent to the definitions given in section 3.2.2 because the boundary crossing connections between nodes in different subgraphs appear only at the leaf level (i.e. in the base graph). The problem with compound graphs is that not all connections between subgraphs have an according edge at the leaf level. We call these edges **coarse edges**. In a sense, these coarse edges are inaccurate or vague connections if they exist only on an abstract level between subgraphs, without reference to the most detailed nodes (i.e. leaves). We essentially lose information with this way of modelling, since it is not specified *where exactly* this edge leads into the subgraph (see also Sect. 4.2.4). This has tremendous implications for path finding across different hierarchy levels: we do not know where an abstract path continues in the subgraph, thus cannot relate it to a concrete path running inside the subgraph.

Nested graphs [PL94] are yet another formalisation for hierarchical graphs, based on simple equations:

Definition 3.2.5 (Hierarchy as Nested Graphs). We can define a hierarchy as a set of graphs, i.e. equations of the form $G = (N, E)$. Different graphs $G_i = (N_i, E_i)$ and $G_j = (N_j, E_j)$ can be related by means of defining a nesting $G_i := n_j$ for an $n_j \in N_j$.

The definition of nested graphs is appealing from a pragmatic point of view: in practice, there are often a set of independent graphs to be connected (instead of just one base graph to be split). This assumption is realistic for geospatial networks (cf. Sect. 4.1.1). Compared to the previous definitions, nested graphs allow only edges between nodes in the same graph, that is siblings in the hierarchy. For nested graphs, in turn, the same problem applies as for compound graphs: there are coarse edges, on an abstract level, which have no corresponding nodes as leaves.

3.2.4 *Important Questions for Spatial Applications*

Practical Stance

Remember that our primary attention is not focused on a theoretical discussion of various formalisms for defining hierarchical graphs. We are instead more interested in dealing with hierarchical graphs from a *practical* viewpoint. Therefore implementation issues are the first priority in our considerations. Our goal is to model networks from the spatial domain – first and foremost indoor environments – by means of hierarchical graphs.

First of all, it has proven useful for indoor environments to choose a special kind of graph for modelling, namely a **multigraph**:

Definition 3.2.6 (Multigraph). A multigraph $G_m = (N, E_m)$ has a set of nodes N and a **multiset** of edges. For each pair of nodes $n_1, n_2 \in N$, there is a separate set of edges $E_{n1 \times n2} \in E_m$. Each edge $e \in E_{n1 \times n2}$ is called **multiedge**.

This definition comprises a generalisation of graphs, allowing several parallel edges between a pair of nodes. Each multiedge is distinguishable from the others. It is convenient to use these multigraphs instead of ordinary graphs as a basis for forming a graph hierarchy, because they are more expressive. In the next chapters, it is shown why this modelling makes sense for geospatial networks, especially indoor environments. In respect of the hierarchy, a question emerges to reflect on:

> *How can a hierarchical graph structure be constructed, stepwise?*

Implications for System Design

Related to this question is also the point what a hierarchical data structure should look like and which basic operations should be supported for its construction and further manipulation. Therefore our definition should orient towards usability and provide easy-to-implement operations on a hierarchical graph. These aspects are covered in Chapter 4 in more detail. We especially need a consistent, versatile representation of a hierarchical graph structure which can be updated and manipulated in many different ways.

Particularities of Spatial Networks

For this purpose though, one has to become aware of one important fact: dealing with (geo)spatial networks, the above made assumptions for obtaining a hierarchical graph are *not* realistic; they often do not hold:

1. we usually do not have a hierarchical structure *a priori*.

2. neither do we have *one* comprehensive base graph from which the hierarchy can be constructed in a *top-down* manner. The starting point is more often a set of ordinary flat graphs which are unrelated (see Sect. 4.2).

Top-Down vs. Bottom-Up Construction

This view however goes beyond the simple hierarchisations in which one base graph is recursively split into different subgraphs. The simple hierarchisations used in the previous definitions do not apply in this specific situation where we lack a single base graph to start from. In fact the opposite is the case: instead of partitioning one large graph into pieces (*top-down* construction), one has to assemble this graph first from a number of smaller graphs (*bottom-up* construction). Each of these smaller graphs appear as independent fragments in the large graph. The main focus of this approach is on **integration**.

Need for Extensions

The flexible construction and maintenance of a consistent hierarchy is a little more sophisticated than in the previously discussed cases. We need to understand hierarchical graphs in a broader sense so that they can be constructed in more difficult situations as well, i.e. in absence of a base graph. How can, for example, different spatial networks be integrated seamlessly into a larger system if this system involves many different spatial networks (a typical case in an urban environment)? This problem is tackled in the next chapter, where the idea of using hierarchical graphs as a geospatial world model is explained.

3.3 HIERARCHICAL PATH FINDING

Shortest path algorithms in graphs are among the most well-studied problems in Computer Science and Artificial Intelligence. Thousands of scientific articles have been published that build on Dijkstra's note from 1959 [Dij59]. Variants of this algorithm are often used in practice. They work rather efficiently, that is in $O(n \times \log n)$ time (with n being the number of nodes in the graph) given a smart use of advanced data structures like Fibonnaci heaps [FT87].

However, in very large and complex systems, such as Geographic Information Systems (GISs) for traffic and traveller information, there has been a need for improving the performance for queries over graphs of massive size. Pragmatic considerations speak in favour of using hierarchical graphs as underlying navigation model: for this purpose, the large graphs have been partitioned into a set of smaller fragments. The organisational structure behind this was a hierarchy. Experiments conducted e.g. by Jing et al. [JHR98] and Shekhar et al. [SFG97] could show a gain in the processing time of shortest path queries by using hierarchical graphs instead of ordinary graphs.

We now want to present some of the different algorithms for hierarchical path finding and their general idea. Roughly, hierarchical path finding algorithms can be subdivided into three main categories:

- classic refinement search,

- hierarchical A*,
- hierarchical optimisation with (partial) materialisation of costs.

Each of the three families of algorithms has its particular strengths and weaknesses. In order to compare them, we consider the following set of criteria: *optimality* states whether the algorithm is guaranteed to find an optimal path with respect to a given cost function, *completeness* states whether the algorithm terminates (at all) on different sorts of input graphs, *heuristic* expresses, if necessary, the dependency of the algorithm on a particular heuristic, *pre-processing* explains if the algorithm can be run instantly, or else what form of pre-processing is required, and finally *parallelisation* describes the number of sub-tasks and -paths which can be computed in parallel.

We illustrate the principal ideas of the subsequent algorithms on a simple example:

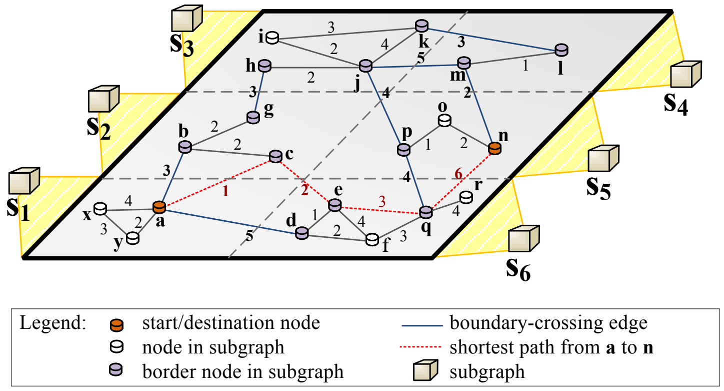


Figure 10: Example for Illustrating Different Methods of Hierarchical Path Finding

3.3.0.1 *Refinement Search*

Refinement search is the most classical form of hierarchical planning. It was proposed in the early days of AI research by Sacerdoti [Sac73]. The underlying idea is very simple:

Basic Idea

It assumes that a problem space (in our case, a base graph) is described at successive levels of abstraction. In Fig. 10, this is just one additional level of subgraphs. The motivation of this approach is to solve problems by looking at their most abstract description first: the algorithm finds a solution (i.e. path) in the simpler abstract description, and then refines it to several problems which have to be solved in the next level of detail, and so on. This process is repeated until a solution is found in the original problem space. The abstract graph where refinement search starts for the concrete example is depicted in Fig. 11 below.

Pros

On the one hand, we have the advantage that refinenemt search pre-selects a number of promising candidates for abstract solutions.

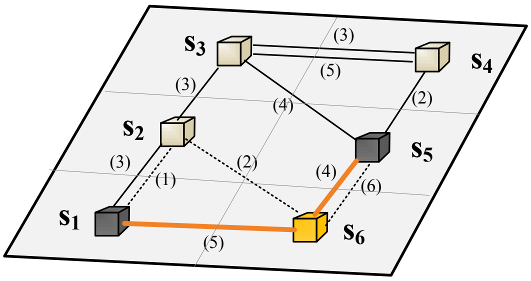


Figure 11: Refinement Search in an Abstract Graph

Unpromising alternatives are pruned already in the early stages of search. In practice, this can accelerate search considerably because when we exclude abstract nodes, we exclude whole subgraphs at once. Besides this, solving partial problems at one particular abstraction level can be done in parallel. These solutions need to be combined in the order of the coarser solution.

Cons

On the other hand, refinement search relies on the fundamental property that search is conducted **monotonically**: that is, once an abstract solution (path) has been determined, it is refined immediately (in the example of Fig. 11, the abstract solution is s_1, s_6, s_5). This choice influences all subsequent steps. They cannot be changed later on, when it should become apparent that there is a better solution for the problem (like the path a,c,e,q through s_2). So essentially, there is *no backtracking* from a finer abstraction back to a coarser abstraction. This way, refinement search cannot guarantee finding optimal solutions: there is a trade-off of reduced processing time against the quality of solutions. For practical purposes, the found solutions could still be good enough. Refinement search instead makes use of a heuristic which assesses costs to abstract solutions. Usually, a heuristic works well if the weight of one is assigned to every abstract edge [HMZM96]. Solutions may well be suboptimal in case that the heuristic costs of an abstract solution (path) turn out to be underestimated. The algorithm's overall performance and thus the quality of the final solution very much depends (besides the heuristic's significance) on the way the problem is decomposed and abstractions are formed. Moreover, refinement search is unfortunately not even complete – there can be certain graphs with corresponding abstractions where the algorithm would need to backtrack because no solution can be found for a subproblem (although the abstract path is valid). An example are subgraphs which are *not* internally connected and therefore cannot be traversed (if the connection from b to g were removed in s_2, for instance). This issue has to be handled in a pre-processing step where the abstractions (subgraphs) of the problem are formed. One could allow for instance only subgraphs which form connected components as valid abstractions. Still, the problem remains unsolved for dynamic cost functions, when costs depend on a particular user group or role (e.g. no traversal of edges typed 'stairs' for persons in a wheelchair or nodes of type 'laboratory' for non-scientists).

3.3.0.2 *Hierarchical A**

Hierarchical A* [HPZM96] is another way of hierarchical path finding. It differs from refinement search in one important aspect: backtracking is allowed. Search is however performed just in the base graph so that there are no advantages of parallelisation whatsoever. The algorithm is therefore not hierarchical in a strict sense; the hierarchy is merely exploited for obtaining a good heuristic. So like its non-hierarchic variant, it uses an admissible heuristic [HNR68] to guide search (by underestimating the remaining distance to the goal node). Unlike refinement search, an optimal solution is found and the algorithm is complete.

In the hierarchical case, this estimate is defined as the shortest distance of the abstraction of the current node to the abstraction of the goal node. It can be calculated from the abstract graphs. Abstract edge weights are set to one, so that they do not overestimate the actual weights in any case. It has proven useful in practice to cache the heuristic function for the same goals. As shown by Holte et al. [HPZM96], hierarchical A* outperforms blind search (i.e. search in the original base graph without hierarchisation), although in some cases only marginally.

3.3.0.3 *Hierarchical Optimisation with Materialisation*

Finally, there is also hierarchical optimisation which makes use of materialisations of partial solutions. These algorithms have been proposed among others by Jing et al. [JHR98] and Shekhar [SFG97]. They provide optimal solutions, but require a remarkable amount of pre-processing for this purpose: First an auxiliary graph is created at every level of the hierarchy, beginning from the most detailed graphs. This auxiliary graph consists of all border nodes of the respective subgraph, their boundary crossing edges, plus abstract edges for pairs of border nodes of the same subgraph if they are internally connected (i.e. there exists an internal path in the respective subgraph). Their weights are thus of the shortest paths between border nodes. These have been previously calculated by a normal all-pairs shortest-path algorithm in the subgraph. An example of such an auxiliary graph is shown in Fig. 12:

Additionally, for every node a routing table is stored. It contains the next node on the shortest path to all other nodes in the same subgraph and the corresponding costs to get there. This table is called 'encoded path view' [JHR98]. It is hierarchical because edge weights of abstract levels depend on the shortest paths at more detailed levels. There is no use of a heuristic. However, with this massive amount of pre-computation, some limitations of the approach become apparent: if, for example, some edge weights can only be determined at runtime (in case a particular query with context information is posed) or the weights change due to some constraints, all preprocessing cannot be done or was in vain. Notwithstanding, some limited possibilities of adjusting paths for changed weights exist in the approach of Jing. But

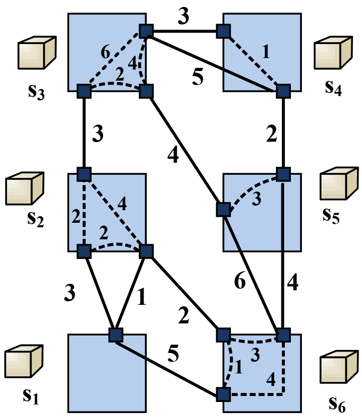


Figure 12: Auxiliary Graph between Border Nodes

with an increasing number of changes, the preprocessed structure becomes more and more inappropriate for queries.

3.4 SUMMARY

For the reasons laid out in Section 3.1, graphs can be extended to hierarchical graphs; the latter provide additional means for grouping/clustering nodes and edges. Knowledge representation with hierarchical graphs is consequently more rich and diverse compared to ordinary graphs. There are a couple of different definitions for hierarchical graphs in literature. They have been discussed in the preceding sections, along with the main criteria for comparing them. Summarising, the additional expressiveness of hierarchical graphs stems from the (repeated) clustering or partitioning of a base graph into different subgraphs. This can be achieved by applying a particular decomposition for this base graph.

Two classic arguments speak in favour of using a hierarchical graph representation:

1. scalability and efficiency when dealing with very large graphs (principle of divide and conquer),
2. additional value for *human-friendly* manipulation and navigation in large graph structures (even for smaller graphs, adding meaningful clustering information can be helpful).

   In addition to these arguments, there is a novel aspect to consider:

3. **integration** of spatial networks.

Hierarchical graphs are not only a convenient means/heuristic for accelerating the computation of shortest paths in large graphs or a better visualisation of these large graphs. The inverse case is also possible: different (sub)graphs can be merged together to compose a larger graph. The next chapter explains how hierarchical graphs can be additionally used as a common basis for the integration of heterogeneous spatial networks.

Part II

CORE DESIGN WITH DATA STRUCTURES AND ALGORITHMS

4 CONCEPTUAL DESIGN OF A HIERARCHICAL GRAPH SYSTEM

In this chapter, the general design issues of a hierarchical graph system are addressed for a practical point of view together with their theoretical underpinnings. One particular requirement is the integration of different graphs from different domains and data sources (e.g. representing different kinds of spatial networks). It is assumed that the data sources are distributed over the Web. A comprehensive system architecture is proposed for this purpose and a proof of concept implementation is presented, too. The system centres on hierarchical graphs. The notion of a mediator is introduced for handling the information flow between different elements (graphs) in the system. Hence the overall system can be regarded as a general toolkit for managing a set of heterogeneous, distributed graphs. A versatile set of operations is provided for incrementally creating a hierarchical graph out of flat graphs. Last but not least, different options are explored for incorporating context information into the system architecture (for path finding under varying conditions and constraints).

In Chapter 5, the introduced basic operations are used when applying hierarchical graphs to the concrete domain of indoor environments. Nevertheless, the concepts defined in this chapter are general enough to be easily applied to other domains as well: For example, one can integrate any conceivable kind of networks which can be represented as (hierarchical) graphs, e.g. indoor environments with a street network and/or a railway system, street networks with a railway system, etc.

4.1 TOP-LEVEL SYSTEM ARCHITECTURE

4.1.1 *Rationale: Distributed, Heterogeneous Spatial Networks*

Exemplary Query

Imagine a navigation system which can provide detailed answers to queries like *"How do I get from the lecture in room D1.13 of the university building to my grandfather in room 134 in the surgical care unit of the Red Cross hospital?"*

Why the Query is Difficult to Answer

The basic problem can be boiled down to a shortest path planning problem, which has been well studied for graphs. However, this problem tends to be much more complicated in practice:

Distributed Networks

In the domain of geospatial information processing, one is especially confronted with the aspect of **data organisation**. In order to answer the query one has to determine its scope – which networks are relevant for answering the query? Solving the path problem sketched above involves knowledge of *both* buildings *and* the intermediate road network. Therefore our navigation system has to consider these networks. However, geospatial data (including spatial networks) are usually *distributed*; there is **no central organisation**. Data pertaining to different networks are proprietary. They are usually owned by different companies or organisations: the government which operates the street networks of a city, public transport companies, various airlines and railway companies, the owner of a particular building, et cetera. This means, in our concrete example, that all three spatial networks pertain to different organisations. They are maintained in separate graphs/systems, independent of each other.[1]

Advantages of Distribution

It is, thus, inevitable to integrate these distributed spatial networks to answer domain-crossing queries in the style of the above mentioned query. Even if data were not kept separately for the above reasons, distribution is beneficial from another, practical point of view: it would not be feasible to have all graphs represented in one single system. In a large application, e.g. comprehending the major cities of Europe, the total size of *all* graphs would certainly exceed the memory capacity of a single system. Instead, the graphs should be organised in a way that they are loaded according to need, i.e. whenever a graph is deemed relevant for answering a query. The above mentioned query for example, requires only the networks of one city (a small fraction of the complete system); there is no need to plan a trip across different cities. So we can prune all other networks for the processing of this particular query.

Processing with Hierarchical Graphs

Distribution is, by virtue of these practical considerations, a central aspect to be taken into account. Among others, it also affects the design of a suitable path finding algorithm: we can only apply a classical graph algorithm on an individual graph. For composing a path across different networks, several partial results need to be combined. Note that the partial results can only be locally optimal – they do not consider the other parts. However, there is also an advantage: they can be potentially computed in parallel given that the graphs are distributed. We can make use of a family of algorithms and techniques specifically designed for hierarchical graphs. They are more sophisticated than in the flat case. Examples include refinement search [HMZM96] or hierarchical planning [Sac73]. The details of these hierarchical path finding techniques are laid out in Sect. 3.3. In Sect. 5.5, we illustrate the basic concepts on the concrete example of indoor environments.

Route Descriptions

Second, even if we employ a suitable path finding algorithm, the resulting paths still need to be *explained* in a way that is comprehensible to people (e.g. in form of verbal instructions and/or graphics).

[1] only large commercial companies can afford to collect vast amounts of data over years and bring them together with great effort.

This is a problem on its own. It is important to give human wayfinders guidance and lead them along a path. Plans in form of qualitative route descriptions are also appealing from the perspective of data owners: They may not be willing to reveal all details of a network (i.e. one should not be able to reconstruct the network by posing a large number of overlapping queries and subsequent path integration). Because of the substantial differences of individual domains (indoor vs. outdoor, public vs. private transport, etc.), specific techniques might be applied for each kind of network. Giving practical path descriptions is more sophisticated than just path finding, although it is very specific for the kind of network considered.

Status Quo

For the time being, a universal navigation system for answering such domain-crossing queries does not exist. Diverse factors are responsible for this:

Considering each system on its own, specialised solutions are available, e.g. for automobiles or other kinds of outdoor spatial networks. However, integration between the different systems is lacking for the most part (due to the inherent diversity and distribution of data discussed before).

Another reason is that pedestrian *indoor* navigation is still a major research challenge. It seems as though indoor environments cannot be handled just in the same way as outdoor environments. To date there is no unique, generally accepted representation of an indoor environment, but a variety of different representations prevail (see Sect. 2.1). Under these circumstances, integration is quite a demanding task.

Consequently, none of the existing systems is capable of planning *across* all three different networks of the example (two of which are indoor environments). For outdoor networks though, there are simple approximations for working around integration problems if a common global spatial positioning and reference system like $WGS-84$ is employed. One can estimate a path in a foreign network only by its Euclidean distance if internal data is not available or accessible. In any case, the core of the problem today lies in this inherent distribution. That is also the reason for plans being mostly restricted to one network and/or domain only.

Online Route Planners

Consider for example existing online route planning systems (a selection can be seen in Table 2): Their architecture can be roughly divided into the three components of user interface, core application system, and digital map data. This three-tier architecture follows a common pattern for Web-based information systems. These route planners are specialised, but standalone systems relying mostly on a flat graph structure. Especially, they are not designed to work together with other route planning systems.

Preliminary Analysis

It is striking that different Web pages essentially use the same application core. Furthermore, comprehensive map data is difficult to find – only few companies provide digital map data. The interest of data owners is to protect their data. All these factors impede the

User Interface / Web Page	Core Application System	Provider of Digital Map Data
map24.com	MapSolute	Europa Technologies, Tele Atlas, NAVTEQ
www.meinestadt.de	mapsuite	Tele Atlas
Route Planner of *aral.de*	MapPoint	NAVTEQ
www.maps.google.com	Google Maps API	Tele Atlas

Table 2: Architecture of a Couple of Online Route Planners [Was07]

design of a generic system. In an ideal architecture the same kind of components would adhere to the same interface. They would be replaceable, so we would consequently have a much more flexible architecture.

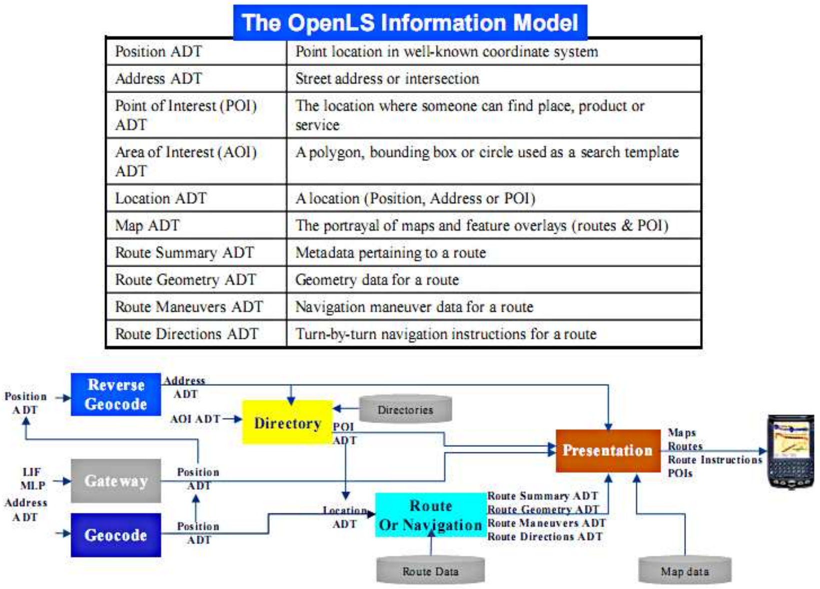
OpenLS – Towards Standardisation

In recent years the Open Geospatial Consortium (OGC) has put some major efforts into the integration of geographic Web services for navigation and location-based services, predominantly with the OpenLS initiative (see [Ope]). The OpenLS is, as the name suggests, a set of open standards that specifies common interfaces for this purpose. Specifically, Part 6 is dedicated to the specification of navigation services and the exchange of location information (the details are shown in Fig. 13). It is to a large extent inspired by the domain of car navigation, for which the most elements are specified. But there has been growing interest for human outdoor navigation as well with the availability of mobile devices and upcoming location-based services, so that walkways and means of public transport are also taken into account. In the specification, different requirements are stated by means of use cases. They are then converted into an XML schema with abstract data types. This schema is destined to be used as a standard protocol between different components implementing a navigation service. However, the internal processing has to be implemented for each component individually.

Review

Although the approach certainly reveals good points, it is somewhat dominated by the theme of car navigation or, more generally, outdoor scenarios. Nonetheless, it is worthwhile investigating to what extent the standard can be adopted for the purpose of indoor navigation. This cannot be done in an ad hoc way, but requires further adjustment:

The defined concepts do not yet cover indoor environments. However, we could use the generic, high-level concepts in the specification and extend them. To give a concrete example, the `RouteSegmentCategoryType` would have to be extended to include rooms, corridors, and other indoor structures. Moreover, the `RouteCostModelType` and `CostCriteria` would have to be subclassed in order to define specific criteria applying to indoor environments. Ideally, this would result in an ontology/taxonomy for the elements of a building specifically tailored for navigation. Depending on the level of detail, this extension could probably be realised with a manageable effort.



The OpenLS Information Model

Position ADT	Point location in well-known coordinate system
Address ADT	Street address or intersection
Point of Interest (POI) ADT	The location where someone can find place, product or service
Area of Interest (AOI) ADT	A polygon, bounding box or circle used as a search template
Location ADT	A location (Position, Address or POI)
Map ADT	The portrayal of maps and feature overlays (routes & POI)
Route Summary ADT	Metadata pertaining to a route
Route Geometry ADT	Geometry data for a route
Route Maneuvers ADT	Navigation maneuver data for a route
Route Directions ADT	Turn-by-turn navigation instructions for a route

Figure 13: OpenLS Information Model [Ope]

Can OpenLS be Adopted for Indoor Environments?

However, there is a more acute problem. The style of giving instructions in indoor environments is usually more complicated and does not necessarily match with the simple turn-by-turn instructions suited for car navigation. This fact is, among others, backed up by Hansen et al. [HKR06], who have extended the OpenLS specification for providing cognitively more adequate route descriptions, e.g. with a special technique spatial chunking. The extension is ergo called 'Cognitive OpenLS'. A further, related point of criticism is that the semantics of navigation can only be covered to a part with such a schema specification. A proper ontology as means of knowledge representation would be a more adequate solution.

It therefore makes sense to reconsider the suitability of OpenLS only after having investigated the characteristics and complexity of route instructions for indoor environments. So the question is deferred to future work.

4.1.2 *Hierarchical Organisation via Mediators*

Building the Foundations for a Meta Navigation System

With a hierarchical graph architecture, however, the lack of integration between separate networks can be overcome. In this chapter, the necessary foundations are explained for realising such a meta navigation system (which plans across the borders of a single domain/graph). The goal is to provide a system capable of managing several different networks side by side. These networks are distributed in a physical sense: they can be separately stored on different, possibly remote machines. For the system design this means that there is not a single instance controlling the entire data. Rather, we stick to the principle of '*divide and conquer*': each network/graph can be seen as an independent unit encapsulating its own data. There is also a component for each graph that manages queries to it. Hence, we are dealing with a distributed architecture – each component is responsible for a certain part of the overall work.

Mediator for Coordinating Distributed Networks

But this alone is not enough: each component operates without detailed knowledge of any other parts. For linking together different networks (e.g. to answer queries *across* different networks), a central component is required.[2] We call this coordinating component the **mediator**. The mediator comprises an integrated computational representation of distributed networks. Each of these networks can be accessed via an Unique Resource Identifier (URI). Moreover, the mediator plays a central role in query processing: the mediator accepts queries from users (or other systems), then delegates subtasks to the corresponding components (networks), composes the partial results, and finally delivers the result to the user/system. A prototypical example of this high-level architecture is illustrated in Fig. 14. As a prerequisite, the partial results of the single graph systems should be compatible for they have to be combined. The final result delivered by the mediator is a concatenation of such partial results with linking elements between networks (see Fig. 14).

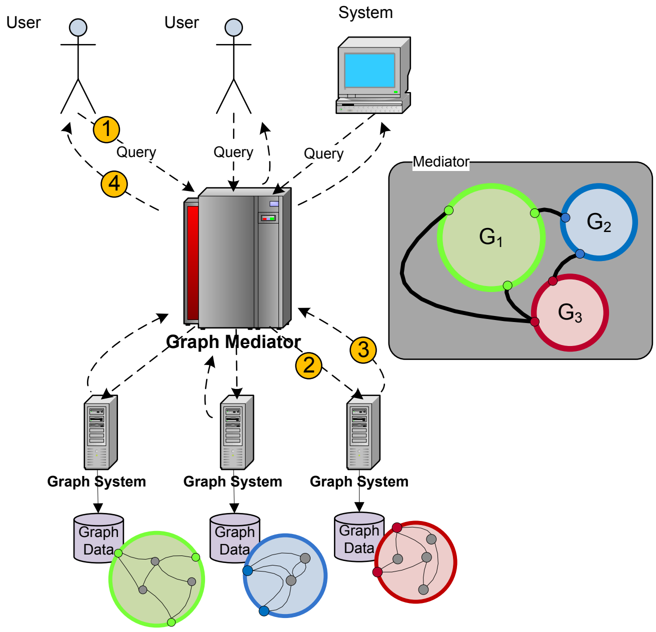


Figure 14: Hierarchical Graph System with a Mediator

Looking inside a Mediator

In a mediator, we internally employ an abstract multigraph representation: each managed network/graph is abstracted to a node, and each individual connection across different networks is abstracted to

[2] In this sense, our approach is not truly distributed (only the graphs are). Alternatively one could let different graph systems communicate *directly* with each other, without the relay through a central component. However, in this constellation search would correspond to a blind (i.e. uninformed) search in a large graph. The advantage of a hierarchy is that it can be used as an heuristic for guiding search. We therefore favour the hierarchical approach here.

an edge.[3] For seamless navigation between different networks, it is essential to know where these connections lead into the respective networks (i.e. to which nodes). That is why a set of **border nodes** are published by each network before the latter is added to the mediator. In a way these border nodes can be seen as public interfaces of a network: they state where external connections to other networks are possible. A formal definition of border nodes is given in the following, in Sect. 4.2.4. Note that before adding a network to the mediator, it also makes sense to check whether the network is *internally* connected.[4] If this is not the case, the graph could be split into its connected components, each becoming a node in the abstract graph of the mediator.

Proof of Concept Implementation

In the work of Martin Wassermann [Was07] supervised by the author, such an architecture has been prototypically implemented for a two-level hierarchy with one mediator as a proof of concept. For technical details regarding the implementation in `Java` (such as establishing communication between mediators and networks via `RMI`), the interested reader may consult the diploma thesis of Wassermann.

Extension to Multiple Levels

In order to be extensible to multiple levels, the mediator needs to be transparent in the following sense: it has to behave like a normal graph system to the outside, offering the same interface as any ordinary graph system which connects to the mediator. This way, different mediators can be integrated into another, high-level mediator on top of them. The system on the top right of Fig. 14 could thus again be a mediator which delegates its queries to the underlying networks and mediators. The desirable extension into a multi-level architecture is sketched below in Fig. 15:

Information Hiding via Mediators

Each mediator is responsible for coordinating a certain number of networks. The internal details of a network remain however under the control of the respective network. This is a classical example for the principle of **information hiding** : each mediator asks its graph systems, not knowing whether they are recursively subdivided by another mediator into different graph systems. Although the mediator has no internal knowledge of its governed networks, it has instead an overview of connections between different networks (see Fig. 14). This knowledge is important for composing the partial results of the individual networks. So communication is only required between a network/mediator and its superordinate mediator (and not directly between different 'sibling' networks). In general, a top-level mediator has to wait for the deepest branch to finish (which takes the longest time for processing, cf. Fig. 15) until the final result can be composed and delivered. There is on the other hand the advantage

[3] Note that there can be multiple connections between two networks (e.g., an underground station usually has several exits to surrounding streets). Each individual connection is represented by one edge.

[4] This is a useful property for navigation; otherwise a continuous path through the network (between different border nodes) is *not* guaranteed to exist. A more detailed path search inside the detailed graph may find out that the graph is not internally connected. This may entail backtracking in a coarser graph, because of the false initial assumption.

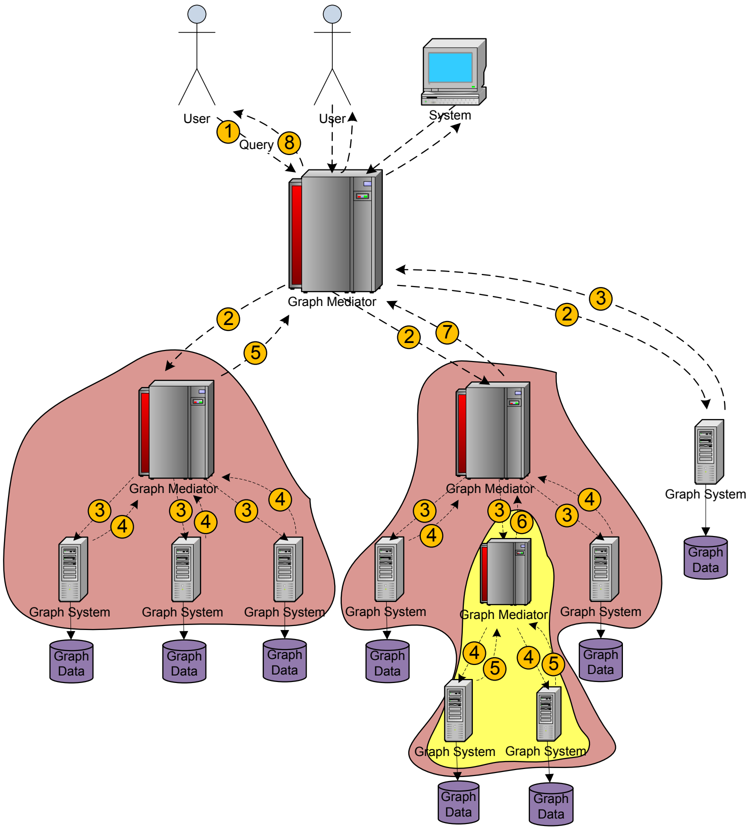


Figure 15: Generalisation to Multiple Levels of Mediators

that computations of different partial results are usually done in parallel.

In summary, the mediator supports the following tasks:

- it manages a set of networks which are distributed in a transparent manner,
- it provides the means for creating connections between different networks,
- it provides an infrastructure for communication and data exchange with the networks,
- it delegates queries to its managed networks and provides an uniform interface for the results.

Now we want to take a step back and highlight the details of such a mediator architecture, above all how the architecture can be realised with hierarchical graphs. The first issue is how cost functions can be generalised as to include context information (cf. Sect. 2.2.5) and the implications for path finding algorithms operating on a hierarchy of graphs.

4.1.3 *Context-Adaptive Path Finding in a Hierarchy*

As discussed in Sect. 2.2.5, different persons may have different preferences, access rights or requirements. This knowledge is subsumed under the general notion of **context information**. We now want to take these different factors into account for path finding, more precisely for adapting paths so that they adhere to all these criteria. Conversely, those paths which do not fulfill the specified criteria should be excluded from the final result. For this purpose, context information needs to be encoded in the graph structure and referred to in the path queries.

The first goal is to provide a convenient means for specifying context information. In Sect. 2.2.5 it has been identified as an important requirement that domain experts and users can easily express context information and related conditions. Here one has to distinguish between a so called **user profile** stating the general abilities, preferences and description of a user on the one hand and on the other hand, a more detailed representation of the underlying concepts of graph elements (including, e.g., the type of nodes and edges and other properties). For the reasons presented in Sect. 2.2.3, the preferred choice was to use a formal ontology language (more precisely OWL[5]) for modelling. Beside this, there is plenty of tool support for creating and managing OWL ontologies, e.g. with free graphical editors like Protégé.[6]

In student work under the direction of the author [Ide06], such an ontology has been implemented (see Appendix A): there are basically three main concepts (representing persons, nodes, and edges in different transport networks) with a set of related properties. For the sake of simplicity, these properties are henceforth referred to as **key-value pairs**. Instances of these concepts describe a particular user or graph element (i.e. node/edge) together with their properties. Graph elements can be associated with properties such as different points of interest (nodes and edges of a certain type etc.) which can be used for determining relevance in a particular context/application. We write $u.k = v$ for an instance u of a user whose property named k has the value v, likewise $e.k = v$ for an instance e of an edge, and $n.k = v$ for an instance n of a node. To get an impression of this ontology, an excerpt of some concepts is displayed in Fig. 16 below: With the help of this ontology, one can now specify simple constraints stating whether traversal through a certain graph element is generally possible/permitted, or only on certain conditions (" do not go through the *laboratory section*, or any *lecture hall* while there are courses" or "do not use *fire emergency doors* in normal situations"). Common-sense knowledge, as shown in the two previous examples, is an important factor for path finding in a realistic application. It thus needs to be taken into account.

The cost functions we are looking for are different from the ones presented in Sect. 2.2.5: Although several factors may play a role in determining whether a certain graph element affords movement, they are not combined by means of numeric operations (as in aggregated

[5] Web Ontology Language by the W3C, see http://www.w3.org/TR/owl-features/

[6] available at http://protege.stanford.edu/

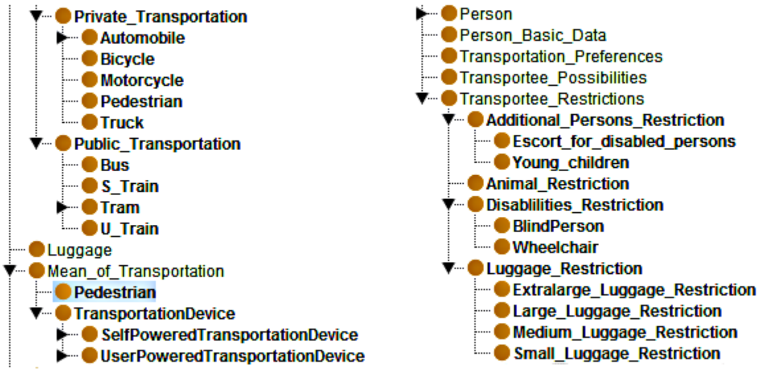

Private_Transportation
Automobile
Bicycle
Motorcycle
Pedestrian
Truck
Public_Transportation
Bus
S_Train
Tram
U_Train
Luggage
Mean_of_Transportation
Pedestrian
TransportationDevice
SelfPoweredTransportationDevice
UserPoweredTransportationDevice
Person
Person_Basic_Data
Transportation_Preferences
Transportee_Possibilities
Transportee_Restrictions
Additional_Persons_Restriction
Escort_for_disabled_persons
Young_children
Animal_Restriction
Disablilities_Restriction
BlindPerson
Wheelchair
Luggage_Restriction
Extralarge_Luggage_Restriction
Large_Luggage_Restriction
Medium_Luggage_Restriction
Small_Luggage_Restriction

Figure 16: Excerpt from the Ontology Modelling Context Information

cost functions), but rather on a symbolic level. As an example, one can state that the cost to traverse a graph element is simply its geometric distance or required time if access is permitted. This is one simple optimisation goal. Determining whether access is permitted can be boiled down to the evaluation of **boolean constraints**[7]. Those can be expressed by making use of the properties defined in the ontology for the graph elements and user profiles (and forming conjunctions): $BC = \bigwedge^{*} (C.k\ \theta\ value)$ with an operation θ like $\geqslant$, $=$, $<$ of integers, *isSubstring* for strings, etc. The key k can be a property from a concept C which is either a graph element or a user profile. Note that keys from different concepts can be mixed in conjunctions. It may happen that a considered graph element N/E does not have a specific key $N.k/E.k$ stated in a constraint BC. We adopt the **open world assumption** in this case, meaning that BC is completely ignored for this graph element.

Let us give a few concrete examples for such constraints: the constraint $(N.type = office) \wedge (U.dayTime > 9am) \wedge (U.dayTime < 8pm)$ would model typical opening hours, $(U.wheelChairFriendly = true) \wedge (E.type = stairs)$ the incompatibility of stairs for persons in a wheelchair, and $(N.type = laboratory) \wedge (U.group = scientist)$ accordingly restricted access to a laboratory. There is a prototypical implementation [Ide07] of these boolean constraints as SWRL rules for the proposed hierarchical graph system.[8] Our decision of using SWRL is yet again based on existing tool support – there is a plug-in for Protégé which facilitates the creation of ontologies considerably.

So much for the specification of boolean constraints. Now let us turn our attention to their evaluation. For the proposed system design there are three different ways of dealing with context information in form of boolean constraints:

- one can evaluate all boolean constraints on different graphs prior to path finding. Nodes and edges which do not fulfill the

[7] yielding the distinct truth values `true` or `false`
[8] SWRL stands for Semantic Web Rule Language. It is a standard recommended by the W3C (see `http://www.w3.org/Submission/SWRL/`).

given set of criteria are filtered out right from the beginning. Path finding is then performed on a reduced, query-specific version of the original graph.

- one can make the design of path finding algorithms more generic, i.e. by providing an interface for injecting boolean constraints into the core part where the next visited nodes are determined. In other words, this enables us to employ different strategies for traversing a graph (not only depth- vs. breath first search, but also based on context information as a parameter). The idea adheres to the strategy design pattern [GHJV95]. Evaluation of conditions then happens on-the-fly (or lazy), while an instance of a path finding algorithm is actually running. This way only the behaviour of iterators over a graph are changed; the structure of the graph remains the same.

- one can ignore the constraints at first and perform path finding as usual. Then, in a post-processing step, the constraints are evaluated: if the resulting path should violate some constraints, the solution can be locally corrected by replanning or be pruned.

The first option has been prototypically implemented by a student under the supervision of the author [Ide07]. It has the disadvantage of working with several copies of the original graph. Such a copy has to be held for *every* query or instance of a path finding algorithm. In a true multi-user system, this circumstance may prove to be counterproductive.

The second option is more complicated. It requires modifying the behaviour of path finding algorithms to some extent, implying a change deeper in the algorithm. However, it is conceptually more elegant: instead of a query-specific *graph*, we obtain a query-specific *algorithm* which traverses the graph (and even a user-specific algorithm adhering to the imposed user constraints). The idea of interchangeable algorithms is in accordance with the strategy design pattern [GHJV95]: it provides a general interface for a certain algorithm/problem and allows one to select from several concrete algorithms (implementing this interface) in a flexible way, at runtime. One can realise different views and traversals of the same graph without actually modifying it. However this approach also has some drawbacks: Apparently it requires the highest implementation effort. In practice, not every available graph system (e.g. from a third party) is designed to provide such a generic interface although it would be desirable. We have nevertheless foreseen this variant and implemented it as part of the mediator-based architecture.

With many constraints the third option becomes, in general, quite inefficient – in the worst case, a complete replanning would be necessary to fix all violated constraints (e.g. if the optimal solution goes through an unaccessible subgraph as a private wing). This problem is alleviated if constraints are considered, too, in the abstract graphs.

Our approach considers not only nodes and edges in one graph, but rather in a hierarchy where abstract nodes can have subgraphs describing their content. This allows us to specify constraints for whole subgraphs ("do not go through the library"), which is very convenient. Otherwise hierarchical path finding is more difficult in the presence of constraints: an abstract path, which is assumed to be traversable through a refined node, could actually turn out to be interrupted (by a closed door, incompatible network or similar constraint). We would thus need to backtrack and replan at a higher level graph. However, this is not foreseen in classic refinement search (see Sect.3.3): it proceeds in a monotonous manner from coarse to fine graphs. One could, instead, use an algorithm for hierarchical optimisation. The only problem is that pre-processed data structures cannot be used anymore – they are static and do not cater for any specific constraints (which are known at query time).

4.2 BASIC OPERATIONS AND CONSISTENT CONSTRUCTION OF HIERARCHICAL GRAPHS

How is the incremental construction of hierarchical graphs carried out in a mediator-based system? This topic is considered here from the beginning:

Initially there is no hierarchical graph in the system, but rather a distributed collection of flat graphs (one for each network) originating from different sources. The assumption is that the graphs are neither arranged in a hierarchy nor related in any other way. Yet we want to bring these graphs together. An abstract hierarchical graph representation (like in the mediator) which links the networks must first be constructed – *incrementally* – from this loose collection of graphs.

It is vital to provide a **set of basic operations** for creating and manipulating a hierarchical graph structure in an incremental manner. In particular, one has to ensure that the hierarchical graph remains **consistent** after *each* operation. This can be done by checking **integrity constraints** before an operation is applied.

From a more general viewpoint, we want to complement the classical view of hierarchical graphs and bring in a novel aspect:

CLASSICAL VIEW (TOP-DOWN). A hierarchy can be obtained by recursively **partitioning** a base graph into different subgraphs (and forming again subgraphs of these subgraphs). We start with *one* base graph containing the whole information of the network structure. This is the classical application of a hierarchy for large networks, for example street networks. The sheer size of these networks necessitates a subdivision in order to better handle them.

NOVEL ASPECT (BOTTOM-UP). A hierarchy can be obtained by **unifying** different distributed networks (which come from different data sources/-providers) and providing additional edges between elements in different networks, so called boundary crossing edges. The approach caters for the integration of different networks, such as **multi-modal** transport and pedestrian networks in an urban environment. One could profit from this representation to enhance multi-modal route planning [Hoc08] (e.g. by using the hierarchy as an heuristic). So far, this aspect of distribution has been neglected to a large degree by the research community. However, we argue that it is important from a practical point of view: it allows for a flexible construction of a hierarchy even if we start with *several* unrelated base graphs. This particular approach is employed in the mediator architecture.

We want to generalise and cover the construction in both cases. Let us now consider the details of the **construction**:

4.2.1 *Hierarchisation via Node Refinement*

First, it makes sense to formally define a basic operation for hierarchisation. In Sect. 4.2.6 it is shown that no other operations (besides the common operations on flat graphs) are required for realising the essential operations of a mediator-based graph system. The following definition serves as a formal basis for constructing hierarchical graphs:

Definition 4.2.1 (Node Refinement Function ρ_{G_r}, Root Graph G_r, Node Refinement Operation $NR(n_c, G_f)$). Let $\Gamma = \{G_1, .., G_n\}$ be a set of **multigraphs**[9] and $G_r \in \Gamma$ a so called **root graph**. A (hierarchic) **node refinement function** $\rho_{G_r} : N_c \rightarrow \Gamma \setminus G_r$ maps a node $n_c \in N_c$ with $G_c = (N_c, E_c) \in \Gamma$ to another graph $G_f \neq G_c \wedge G_f \in \Gamma \setminus G_r$.[10] The image $G_f = \rho_{G_r}(n_c)$ is the **subordinate graph** of n_c and, conversely, the nodes n_c are **superordinate node** of G_f.

A **node refinement operation** $NR(n_c, G_f)$ makes one assignment of the form $G_f := \rho_{G_r}(n_c)$, i.e. $NR(n_c, G_f)$ iff $G_f := \rho_{G_r}(n_c)$.

Static vs. Dynamic Aspects

We conceptually distinguish between the **static function** ρ_{G_r} which describes *all* current mappings of nodes to graphs and the **dynamic operation** $NR(n_c, G_f)$ which adds a concrete mapping of exactly one node to one subordinate graph to ρ_{G_r}. In an intuitive way, the node refinement function ρ_{G_r} can be understood as defining 'nestings' for finer-scaled graphs G_f into superordinate nodes n_c (i.e. the entire graph G_f is collapsed to a coarse representation as one node n_c). These superordinate nodes n_c, in turn, pertain to coarser graphs G_c. From this perspective, the pairs of graphs G_f and G_c are indirectly related via a node refinement function. So we can naturally interpret the node refinement function as a binary relation $\prec$ between two different graphs G_f and G_c. The example illustrated in Fig. 17 shows

[9] Remember from Def. 3.2.6 in Sect. 3.2.4 that multigraphs permit several parallel edges between the same pair of nodes. When referring to graphs from here on, we mean the more general concept of multigraphs. As it turns out, multigraphs are suited for our modelling purposes – whereas simple graphs are insufficient (cf. Sect. 4.2.4 for concrete examples).

[10] The root graph G_r is excluded from the images of ρ_{G_r} to avoid cyclic references.

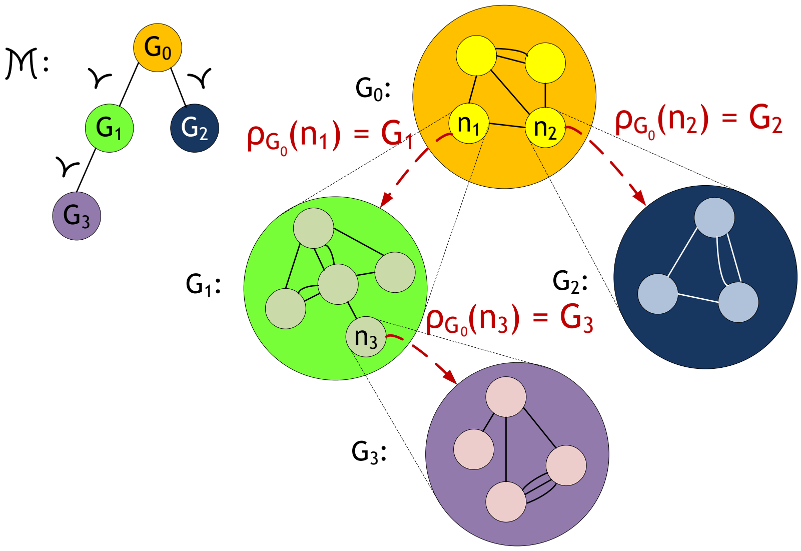


Figure 17: Example for a Hierarchic Node Refinement Function

this connection. Here, the hierarchy is a tree, but what about 'genuine' graphs? This topic is discussed in the subsequent section.

Definition 4.2.2 (Hierarchy Relation $\prec$, Meta Graph $\mathcal{M}$). Let $\Gamma = \{G_1, .., G_n\}$ be a set of multigraphs. The **hierarchy relation** $\prec \subset \Gamma \times \Gamma$ is defined for a pair of graphs $G_f, G_c \in \Gamma$ as:
$G_f \prec G_c$ iff there is a node $n_c \in N_c$ with an image G_f in the hierarchic node refinement function, i.e. $G_f = \rho_{G_r}(n_c)$. G_c is then called superordinate graph and G_f subordinate graph, respectively. With $\prec$, we can define an abstract **meta graph** $\mathcal{M}$ for representing the relations between different graphs in Γ: $\mathcal{M} = (\Gamma, \prec)$.[11]

4.2.2 *Basic Operations for Hierarchical Graphs*

One can distinguish between four basic operations θ for hierarchical graphs:

- insert/remove a multigraph G_i to/from the hierarchy (this corresponds to adding/removing a node G_i in $\mathcal{M}$),
- make/undo a node refinement operation $NR(n_c, G_f)$ (this corresponds to creating/removing an edge (G_f, G_c) in $\mathcal{M}$),
- insert/remove a node n_i in a graph $G_i \in \Gamma$,
- insert/remove an edge e_i in a graph $G_i \in \Gamma$.

Classification

The first two operations concern whole graphs; therefore they affect the meta level of $\mathcal{M}$ whereas the latter two operations are applied locally inside a specific graph G_i. None of these operations can be

[11] The meta graph $\mathcal{M}$ is *not* a multigraph due to the existential quantification in the definition of $\prec$. It is however a directed graph, because the order of graphs in $\prec$ is significant ($G_f \prec G_c$ has a completely different meaning compared to $G_c \prec G_f$). The relation $\prec$ is, in general, **asymmetric** (see Sect. 4.2.3). We thus write (G_f, G_c) for the edge representing $G_f \prec G_c$ in this particular order.

performed arbitrarily in a hierarchical graph. The structure of the hierarchy, even for the local operations on a graph G_i, has to be taken into consideration. We need to investigate the related integrity constraints (i.e. preconditions) for applying each operation. For the four basic operations from above, the results are summarised in Sect. 4.2.5.

Use of $\mathcal{M}$

One can keep record of every node refinement operation (and other operations as well) and construct the meta graph $\mathcal{M}$ in parallel. As we shall see in the next sections, $\mathcal{M}$ is quite useful for checking integrity constraints. For this reason, a node refinement operation $NR(n_c, G_f)$ should additionally add an edge (G_f, G_c) to $\mathcal{M}$. Apart from $NR(n_c, G_f)$, there are also other dynamic operations that have to be taken into consideration:

4.2.3 *Maintaining a Consistent Hierarchy*

The required properties for ρ_{G_r} and $\prec$ are now analysed:

Injectivity of ρ_{G_r}

The hierarchic node refinement function ρ_{G_r} must be **injective** to avoid situations where $G_f \prec G_c$ although G_c has two node refinements for G_f (involving two different nodes of G_c).[12] A graph $G_f \in \rho_{G_r}$ thus always has a *unique* superordinate node. In Fig. 17, for example, no further node may be refined to G_1, because G_1 already has n_1 as superordinate node – any additional operations of the form $NR(n, G_1)$ would have to be rejected, hence. In general, it makes sense to check this condition before applying a new node refinement operation $NR(n_c, G_f)$. This is a first example of an integrity constraint.

Surjectivity of ρ_{G_r}

Moreover, ρ_{G_r} needs to be **surjective**: there may not be graphs $G_f \notin \rho_{G_r}$ – except for the root graph G_r – which have *no* corresponding superordinate node in another graph.[13] Intuitively, every graph apart from the root has to be linked into the hierarchy; there are no 'loose' graphs whatsoever.

This implies that the hierarchic node refinement function ρ_{G_r} must be a **bijection** (that is, a one-to-one correspondence). We impose further conditions ensuring consistency also on the relation $\prec$:

Properties of $\prec$

By definition, $G_f \neq G_c$ for all mappings $\rho_{G_r}(n_c) = G_f$ in a node refinement function. The relation $\prec$ is, as a result, **irreflexive**. Furthermore, it does **not** make sense that the relation $\prec$ is **symmetric** (from $G_f \prec G_c$ we would then conclude that $G_c \prec G_f$). This would be contrary to the purpose of the relation $\prec$, which is to impose an **ordering** on graphs (when a graph G_f has a superordinate node n_c in another graph G_c, it is considered smaller than G_c).

[12] Generally, a function f is called injective iff $f(a) = f(b) \Rightarrow a = b$.

[13] a function is called surjective iff for all images y, an x with $f(x) = y$ exists.

Although the relation $\prec$ is **not transitive** per se,[14] we can nevertheless define an artificial extension $\prec^\star$ for the **transitive hull** of the hierarchy relation. The desired ordering on graphs can be obtained by means of the transitive hull $\prec^\star$.

Implications

Before performing another hierarchic node refinement operation $NR(n_c, G_f)$, one needs to make sure that the resulting hierarchy remains a linear ordering (in the above sense, w.r.t. $\prec^\star$). Otherwise the operation $NR(n_c, G_f)$ has to be rejected in general because it would lead to an inconsistent state of the meta graph $\mathcal{M}$.

The following situations describe, importantly, inconsistent states of the meta graph due to a lack of order among graphs:

1. there are cycles of the form $G_f \prec \ldots \prec G_f$,
2. a graph G_f has two (or more) different superordinate graphs G_{c_1} and G_{c_2}, i.e. $G_f \prec G_{c_1}$ and $G_f \prec G_{c_2}$.

Illustration

This means that, for the concrete example of Fig. 17, one is stuck: no more nodes may be refined to a graph until a new graph has been added. Since the application of a node refinement operation $NR(n_c, G_f)$ can be interpreted as adding an edge $G_f \prec G_c$ in the meta graph (written as (G_f, G_c)), there is an intuitive way to check its consistency with the meta graph:

Definition 4.2.3 (Consistent Meta Graph, -Operation). Let $\Gamma = \{G_1, .., G_n\}$ be a set of multigraphs with a hierarchy relation $\prec$ and $\mathcal{M} = (\Gamma, \prec)$ the corresponding meta graph. $\mathcal{M}$ is **consistent** iff $\mathcal{M}$ is a tree.
An operation θ is **consistent** iff the following condition holds:
$\mathcal{M}$ is consistent $\Rightarrow$ $\mathcal{M}'$ after applying θ is consistent.

Consistent Node Refinement Operations

A node refinement operation $\theta = NR(n_c, G_f)$ for $G_c, G_f \in \Gamma$ is therefore consistent iff $\mathcal{M}' = (\Gamma, \prec \cup (G_f, G_c))$ remains consistent. However, the first basic operation, namely adding a graph G_i to Γ, may not be applied alone. The reason is that ρ_{G_r} would not be surjective after the operation and thus $\mathcal{M}$ not consistent. The operation must be done *together* with a node refinement operation concerning the graph G_i, to ensure that $\mathcal{M}$ remains a tree. So it is in fact a composite operation (see Sect. 4.2.5).

Other Operations

Likewise, we may only remove a node n_i in G_i if there is no subordinate graph $G_f = \rho_{G_r}(n_i)$ (for the same reasons). If the graph G_f *and* the node n_i were removed simultaneously we could also remove G_f from ρ_{G_r} and consequently the edge (G_f, G_i) in $\mathcal{M}$. This is another composite operation (which is consistent, however). With the tree property of $\mathcal{M}$, we can finally complete the definition of hierarchical graphs:

Definition 4.2.4 (Root, Leaf, (k-level) Hierarchical Graph). Let $\Gamma = \{G_1, .., G_n\}$ be a set of multigraphs with a hierarchy relation $\prec$ and a **consistent** meta graph $\mathcal{M} = (\Gamma, \prec)$.

[14] we cannot conclude $G_f \prec G_i$ from $G_f \prec G_c$ and $G_c \prec G_i$ because of the injectivity property of ρ_{G_r}.

A graph G_r in Γ is called **root** iff it has no superordinate graph. Conversely, a graph G_l in Γ is called **leaf** iff it has no subordinate graphs. A node $n \in N$ in a graph $G = (N, E) \in \Gamma$ is likewise a **leaf** iff it has no subordinate graphs. For a graph $G_i \in \Gamma$, let k be the length of the path in $\mathcal{M}$ from the root of G_i to G_i. Then G_i is called **(k-level) hierarchical graph**.

4.2.4 *Consistent Path Finding over Multiple Levels*

So far we have defined hierarchical graphs to establish a structural correspondence between graphs at different levels of the hierarchy. This correspondence is however insufficient in practice. The defined graphs are not yet feasible for **path finding** across different levels. What is still missing is the information that links together graphs of two different levels in the sense of having coherent paths:

Motivation for another Form of Consistency

Let ρ_{G_r} be a node refinement function and $\rho_{G_r}(n_c) = G_f$ with n_c as a node in G_c. Assume that a graph algorithm traverses G_c in search of a shortest/fastest path. It enters the node n_c through an edge e_s and then leaves n_c via another edge e_e. This is *one* elementary step. Now one could profit from the graph G_f – being a more detailed representation of n_c – to give a more precise path between e_s and e_e. In general this path consists of several steps (some intermediate nodes in G_f). The sequence of nodes in G_f can be given in place of the atomic step 'go through n_c' as a more detailed description. For this, we need to know where the corresponding path in G_f starts and ends. This exactly is the problem: the edges e_s and e_e in the coarse graph G_c are in a way indefinite, for it is unknown *where* they lead into the detailed graph G_f. They are only explicit edges in the coarse graph G_c while in fact these edges should be at multiple levels. Otherwise, we cannot switch between different abstraction levels during path finding.

Example

To clarify this idea, let us consider a concrete example: Fig. 18 extends the previous hierarchic node refinement function given in Fig. 17. A hierarchical graph is depicted which models a building at several levels of detail. The root graph, G_0, represents the complete building. Its nodes (among whom are n_1 and n_2) represent constituent floors. The edges of G_0 represent individual staircases that connect different floors. At this point it becomes apparent why we need multigraphs – namely for modelling a set of parallel connections between two floors.

In particular, we are now interested in knowing where the two edges e_s and e_e leading/incident to n_1 *continue* in the subordinate graph G_1. These are, from a practical viewpoint, the transition points from one network to another network (between two floors in this example).

Border Nodes

For this purpose, we need to look into G_1 and determine two corresponding nodes, n_s and n_e (the two rooms/corridors/sections on the floor G_1 where the respective stairs are leading to). We call these special nodes **border nodes**. They are the start- and end points of the more detailed interior path through G_1 (illustrated as colored nodes

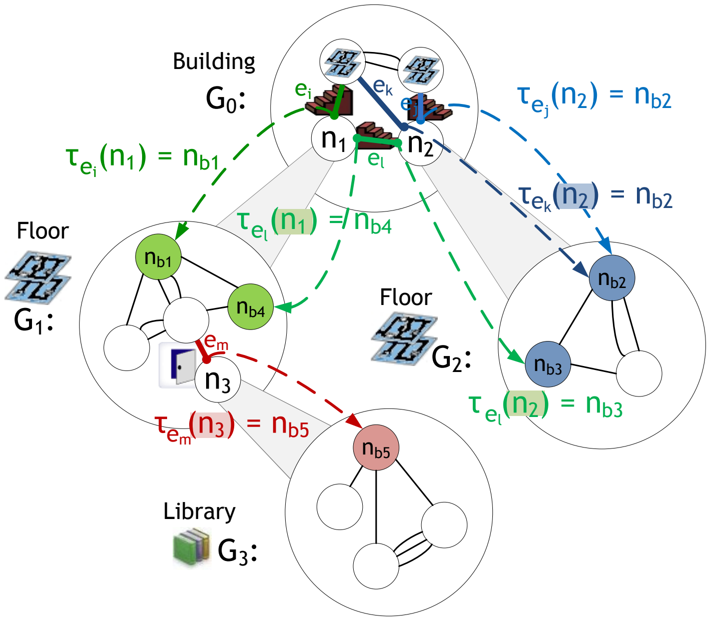


Figure 18: Coherent Paths across Multiple Levels of a Building

in Fig. 18). In general, the nodes of G_1 represent rooms, corridors or different sections within the floor. The node n_3 for instance, represents a separate library section on this floor which has its own graph G_3 associated as a detailed model. Edges in G_1 and G_3 represent traversable openings on common boundaries, such as doors. Since there could be more than one connection between two rooms/corridors/sections, multigraphs are, again, the preferred choice. We examine the details of modelling indoor environments with hierarchical graphs in the next chapter.

Back in our abstract formalisation, we need to relate for all mappings $\rho_{G_r}(n_c) = G_f$ within a node refinement function the specific start and end nodes n_s, n_e of interior paths in G_f with the edges e_s, e_e in G_c incident to n_c. This allows us to substitute an abstract path through a node n_c with an interior path through the subordinate graph G_f. It therefore makes sense to specify another kind of mapping for linking paths at different levels:

Definition 4.2.5 (Path Linking Function τ_{e_c}, Border Nodes, Path Linking Operation $\mathrm{PL}(e_c, n_a, n_b)$). Let $\Gamma = \{G_1, .., G_n\}$ be a set of multigraphs with a node refinement function ρ_{G_r}.
For every edge e_c in a graph $G_c \in \Gamma$ we define a **path linking function** $\tau_{e_c} : \mathcal{N} \to \mathcal{N}$ where $\mathcal{N}$ is the set of all nodes in all graphs in Γ, i.e. $\mathcal{N} = \{n \mid n \in N \text{ with } G = (N, E) \in \Gamma\}$:[15]

$$\tau_{e_c}(n_c) := \begin{cases} n_b \in N_f & \textit{if } n_c \in N_c \textit{ and } n_c \textit{ is incident to } e_c \textit{ and} \\ & n_c \textit{ has a subordinate graph } G_f = \rho_{G_r}(n_c), \\ \text{undefined} & \textit{otherwise.} \end{cases}$$

The assigned node n_b in the subordinate graph G_f is called **border node**. The function τ_{e_c} is *partial* because all non-incident nodes have

[15] The general definition with $\mathcal{N}$ is convenient because we need to extend the function later.

no corresponding border node as image.
The related dynamic operation is called **path linking operation** $\mathsf{PL}(e_c, n_c, n_b)$. It assigns for an edge e_c in G_c and a node n_c a border node n_b in G_b (i.e. $\tau_{e_c}(n_c) := n_b$) if n_c is in G_c and incident to e_c and there exists a subordinate graph $G_f = \rho_{G_r}(n_c)$ and $G_b = G_f$, i.e. $G_b \prec G_c$.[16]
The set of *all* path linking functions τ_{e_c} with $e_c \in E_c$ and $G_c = (N_c, E_c) \in \Gamma$ is denoted as $\mathcal{T}$ without index.

A handful of exemplary path linking functions, $\tau_{e_i}, \tau_{e_j}, \dots \tau_{e_m}$ are depicted in Fig. 18 above. In particular, it is possible to assign the same border node to two (or more) different edges (like n_{b2} in Fig. 18 for the blue edges e_j and e_k incident to n_2; the two edges represent stairs that lead into the same room on the floor G_2). The edge e_l, for instance, links nodes at two different levels: on the one hand, the nodes n_1 and n_2 in G_0 (two different floors); on the other hand, also the nodes $n_{b4} = \tau_{e_l}(n_1)$ and $n_{b3} = \tau_{e_l}(n_2)$ which belong to different graphs, G_1 and G_2 (two rooms in different floors).

Implications for NR

For ensuring coherent paths across different hierarchy levels, one has to specify or adapt for each node refinement operation $\mathsf{NR}(n_c, G_f)$ (that is, every new edge in $\mathcal{M}$) the corresponding path linking functions for each edge incident to n_c. However, the path linking function makes common operations like adding/removing edges on flat graphs more complicated for hierarchical graphs: when adding for example, a new edge $e_c = (n_{c1}, n_{c2})$ in a graph G_c, one should keep this in mind. One has to check whether the two involved nodes n_{c1} and n_{c2} have a subordinate graph G_{fi} as a node refinement (i.e. whether a graph G_{fi} exists with $\rho_{G_r}(n_{ci}) = G_{fi}$). If there are such subordinate graphs G_{f1} and/or G_{f2}, additional border nodes n_{bi} need to be specified via two/one additional path linking operation(s) $\mathsf{PL}(e_c, n_{ci}, n_{bi})$.

In summary, the definition of a path linking function has an impact on the basic operations on potentially all hierarchy levels – including also the meta graph $\mathcal{M}$.

Special Case

However, one problem still remains in the face of a true multi-level hierarchy: What happens if we specify a border node $n_b = \tau_{e_c}(n_c)$ which in turn has a subordinate graph? The subordinate graph of the border node would still lack the connection to the concerned edge e_c. An example for such a problematic situation is pictured in Fig. 19: While the red edge e_m to the library node n_3 in the graph G_1 is a regular door, the stairs e_l leading into the floor n_1 also have n_3, the library, assigned as border node. Because of this we need an additional assignment for e_l to a more detailed border node *inside* G_3 (to the green node n_{b6} in the library). The corresponding *recursive* path linking operation $\mathsf{PL}'(e_l, n_3, n_{b6})$ is indicated in Fig. 19 by the lowest, dark green arrow.

Recursive Border Nodes

In general we need to take into account a possible recursion of border

[16] These preconditions ensure correct assignments, so that the resultant function τ_{e_c} is indeed a path linking function.

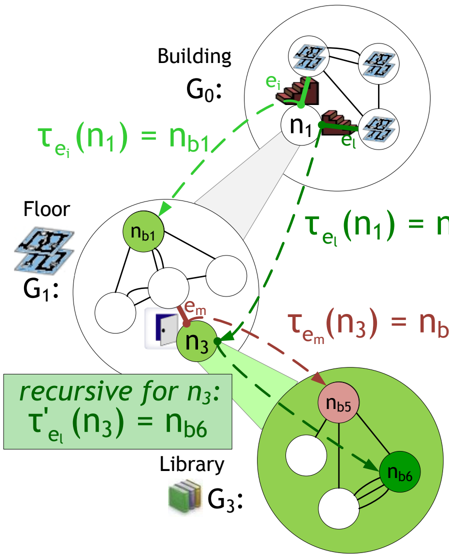


Figure 19: Path Consistency for Border Nodes with Subordinate Graphs

nodes within subordinate graphs of border nodes. This can lead to a cascading *list of border nodes* for path linking functions in hierarchical graphs. We have to extend the notion of a path linking function to meet this special requirement:

Definition 4.2.6 (Recursive Path Linking Function τ'_{e_c}, Recursive Path Linking Operation $\mathrm{PL}'(e_c, n_b, n_s)$). Let ρ_{G_r} be a node refinement function and τ_{e_c} a path linking function. A path linking function $\tau_{e_c} : \mathcal{N} \to \mathcal{N}$ can be extended to a **recursive path linking function** $\tau'_{e_c} : \mathcal{N} \to \mathcal{N}$ in the following manner:

$$\tau'_{e_c}(n_c) := \begin{cases} \tau_{e_c}(n_c) & \textit{if } n_c \in N_c, \\ n_{bb} \in N_f & \textit{if } n_c \in \tau'_{e_c} \textit{ and } n_c \textit{ has a subordinate} \\ & \textit{graph } G_f = \rho_{G_r}(n_c), \\ \text{undefined} & \textit{otherwise.} \end{cases}$$

The nodes n_{bb} are termed recursive border nodes.

The corresponding operation is called **recursive path linking operation** $\mathrm{PL}'(e_c, n_b, n_{bb})$. It assigns for an edge e_c in G_c and a node n_b in G_b a recursive border node n_{bb} in G_{bb}: $\tau_{e_c}(n_b) := n_{bb}$. The operation can only be applied if n_b is in τ'_{e_c} and a subordinate graph $G_f = \rho_{G_r}(n_b)$ and $G_{bb} = G_f$ exists.[17]

The set of *all* recursive path linking functions τ'_{e_c} with $e_c \in E_c$ and $G_c = (N_c, E_c) \in \Gamma$ is accordingly denoted as $\mathcal{T}'$ without index.

Explaining the Recursion

The definition of τ'_{e_c} is recursive. The basis (where $n_c \in N_c$) is equivalent to τ_{e_c}. In short the recursion scheme can be explained this way: Say there is a subordinate graph G_f given as an image $\rho_{G_r}(n_b)$ of the border node $n_b = \tau'_{e_c}(n_c)$. A further border node $n_{bb} = \tau'_{e_c}(n_b)$ has to be specified within this graph G_f and so forth $(\tau'_{e_c}(\tau'_{e_c}(\ldots\tau'_{e_c}(n_c)\ldots)))$, until a leaf node has been reached in the

[17] These conditions ensure merely correct assignments for τ'_{e_c}. For the resultant function τ'_{e_c} to be a recursive path linking function, however, *all* necessary assignments must have been made.

hierarchy which has no more subordinate graphs. This is exactly the fixed point where the function τ'_{e_c} stops 'growing'.

Constructing τ'_{e_c} Incrementally

To obtain a recursive path linking function τ'_{e_c}, a corresponding set of i recursive path linking operations $PL'_i(e_c, n_i^{old}, n_i^{new})$ have to be applied. This happens in a 'coarse-to-fine' manner, beginning from nodes $n_1^{old} := n_c$ in G_c which are incident to e_c. A border node $n_1^{new} := \tau_{e_c}(n_1^{old})$ is assigned to each of these nodes by means of a path linking operation $PL(e_c, n_1^{old}, n_1^{new})$ and a subsequent recursive operation $PL'(e_c, n_1^{old}, n_1^{new})$.[18] We continue with another recursive path linking operation $PL'_{i+1}(e_c, n_{i+1}^{old} := n_i^{new}, n_{i+1}^{new})$ in the same way: n_{i+1}^{old} is substituted by the previous value n_i^{new}, and a new node in the graph $\rho_{G_r}(n_i^{new})$ is specified for n_{i+1}^{new} (if such a graph exists). In case there is no subordinate graph $\rho_{G_r}(n_i^{new})$, the operation PL'_{i+1} is not necessary anymore. We are then finished with the branch starting from n_1^{old}.

Edges between Leaves

Although simple, the rationale behind the recursion is crucial from a practical respect: we need to have sufficient information for *connecting all leaves* in the hierarchy (in terms of Def. 4.2.4), so that we can always give the most detailed paths possible. This leads to an enhanced, even more strict notion of consistency for an operation θ:

Definition 4.2.7 (Path-Consistent Edge, -Operation). Let θ be a consistent operation and e_c be an edge in the graph G_c. The edge e_c is **path-consistent** iff there is a corresponding recursive path linking function τ'_{e_c}. An operation θ is **path-consistent** iff all edges e_c of all graphs $G_c = (N_c, E_c) \in \Gamma$ which are path-consistent *before* the application of θ remain path-consistent *after* the operation, and any new edges are also path-consistent.

Problems

Thus, the path-consistency of all edges can be seen as an invariant for an operation θ. A node refinement operation $NR(n_c, G_f)$ is **path-consistent** iff all incident edges e_c of n_c remain path-consistent after the operation (all other edges are not affected by $NR(n_c, G_f)$ anyway). However, if n_c has some incident edges e_c, then the node refinement operation alone is *not* path-consistent. So we have to supplement such a node refinement operation $NR(n_c, G_f)$ with an according set of recursive path linking operations $PL'_i(e_c, n_i^{old}, n_i^{new})$ for all incident edges e_c. This way we can ensure path-consistency. Further operations which are critical w.r.t. path-consistency are, for example:

- removing any node n_i which has a subordinate graph $G_f = \rho G_r(n_i)$,
- removing/adding any edge e_c incident to a refined node n_c in a graph G_c.

[18] assuming that $PL(e_c, n_1^{old}, n_1^{new})$ is applicable, i.e. there is a subordinate graph $\rho_{G_r}(n_1^{old})$. Otherwise no more recursive path linking operations are needed to make sure that τ'_{e_c} is a recursive path linking function.

4.2.5 *Consistency of the Basic Operations*

Relaxing the Conditions

The previous definitions of ρ_{G_r} and τ'_{e_c} serve as a formal basis for defining consistency criteria for the other basic operations (apart from node refinement) as well: One could require that an operation θ may only be applied if it is **consistent** and additionally also **path-consistent**. However, this requirement may be a bit too rigid in practice, because the flexibility for defining operations would be limited to path-consistent operations only. Note that simple node refinement operations, for instance, were not allowed with this rigid notion of path-consistency. We may want to build some complex operations Θ from a sequence of basic operations $\theta_1 \dots \theta_n$. So it is reasonable to relax the conditions of consistency for composite operations: a composite operation Θ as a whole may well be (path-)consistent, without its constituent operations being necessarily (path-)consistent. In other words, the intermediate states of a composite operation are allowed to be inconsistent as long as the final outcome is (path-)consistent again. However caution is required insofar as we are dealing with a distributed system – different (sub)graphs can be constructed in parallel. Consequently there may be problems with concurrency, especially if a composite operation Θ is interrupted during an inconsistent intermediate state θ_i. To avoid this problem, composite operations are also implemented as atomic (i.e. not interruptible) operations. This guarantees consistency while constructing the hierarchical graph in parallel.

Now let us have a closer look at the modalities for creating a hierarchical graph system:

Definition 4.2.8 (Hierarchical Graph System $\mathcal{G}$). Let $\Gamma = \{G_1, .., G_n\}$ be a set of **multigraphs**, ρ_{G_r} a corresponding node refinement function, and $\mathcal{T}'$ the set of all recursive path linking functions for all edges in all graphs $G_i \in \Gamma$.
The triple $\mathcal{G} := (\Gamma, \rho_{G_r}, \mathcal{T}')$ defines a **hierarchical graph system**.

Construction of Hierarchical Graphs from the very Beginning

By successively adding new graphs into the hierarchical graph system and establishing connections between these graphs, one can create hierarchical graphs. Typically one begins with an empty system where there are no graphs in Γ except for a root graph G_r. The root graph is, initially, also empty. The pertinent meta graph $\mathcal{M}$ has G_r as sole node. So $\mathcal{M}$ is consistent by definition. Since there are no edges in G_r, one does not need to worry about path-consistency either. Now we want to populate the system by adding new graphs in a sequential manner, creating nodes and edges in an individual graph, etc. These operations are the basis for the further manipulation of hierarchical graphs:

ADD A NODE IN A GRAPH. Let $G_i = (N_i, E_i)$ be the multigraph in Γ before the operation. Then $G'_i = (N_i \cup \{n\}, E_i)$ is the resulting graph. Adding a node is the simplest of all basic operations because it has no side effects: $\mathcal{M}$ remains the same, i.e. $\mathcal{M}' = \mathcal{M}$. So the operation is consistent. It is also path-consistent because it does not change any existing edges and does nor create new edges which could *not* be path-consistent. Ergo, this operation can be applied safely at any time.

MAKE A NODE REFINEMENT OPERATION. Path consistency generally requires that a node refinement operation $\mathrm{NR}(n_c, G_f)$ is accompanied by a corresponding series of recursive path linking operations: if n_c has incident edges e_c, one has to specify a set of recursive path linking operations $\mathrm{PL}'_i(e_c, n_i^{old}, n_i^{new})$ for all such e_c (where $n_1^{old} := n_c$ is the starting point, as laid out before). These individual operations together constitute a composite operation.

Intricacies

Likewise, one has to perform an additional series of recursive path linking operations $\mathrm{PL}'_i(e, n_i^{old}, n_i^{new})$ for all edges e with $n_1^{old} = n_c \in \tau'_e$ and specify a border node $n_1^{new} = n_b$ in G_f. Consider the example of Fig. 20: the edge e_l falls into this second category for the node refinement operation $\mathrm{NR}(n_3, G_3)$, because $n_3 = \tau'_{e_l}(n_1)$; therefore the additional operation $\mathrm{PL}'(e_l, n_3, n_{b6})$ is required.

Final Result

If applied correctly, the sequence of operations $\theta = \mathrm{NR}(n_c, G_f)$, $\mathrm{PL}'_i(e, n_i^{old}, n_i^{new})$ is both consistent *and* path-consistent. The whole operation can fail due to a resulting inconsistency of the meta graph $\mathcal{M}$ or an incorrect or incomplete specification of recursive path linking operations thereafter. Any effects made so far have to be undone in this case. In case of success, an edge (G_f, G_c) is permanently added to $\mathcal{M}$. For composite operations it is particularly useful to see, in case of an error, which of its constituent operations failed.

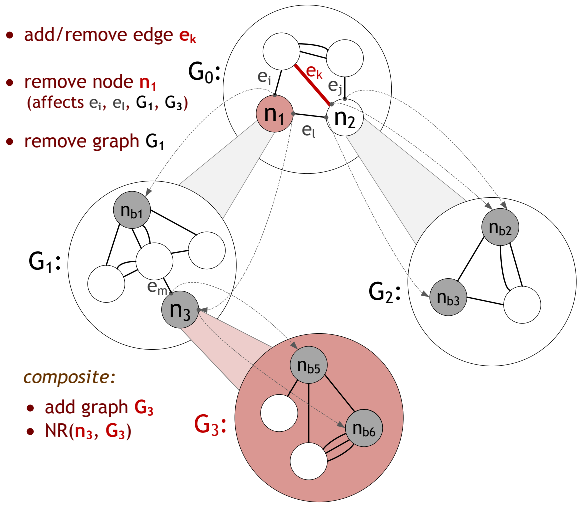


Figure 20: Basic Operations Exemplified on a Hierarchy

ADD A NEW GRAPH TO THE HIERARCHY. Whenever adding a new graph G_i into the hierarchy, one has to mind the following situation: ρ_{G_r} would not be surjective anymore because the new graph G_i is unrelated ($\mathcal{M}$ would have to be a tree spanning over G_i, too). So the operation, per se, is not consistent. In consequence of this, we can only add new graphs G_f one by one to the hierarchy while maintaining consistency. It is not possible to construct a sub-tree involving several graphs and then add this complete sub-tree to the graph system en bloc. Nevertheless, this kind of operation is required among others for bringing graph systems of two mediators together. We define a composite operation for this purpose (see Sect. 4.2.6).

Including Node Refinement

We have to link the new graph G_i immediately to some other graph G_c in Γ, i.e. create an additional edge (G_i, G_c) in $\mathcal{M}$, via an appropriate node refinement operation $NR(n_c, G_i)$. But again we are dealing with a composite operation. It essentially relies on the node refinement operation. As a result all the intricacies indicated above have to be taken into account for ensuring path consistency. In the example of Fig. 20, the graph G_3 is added to the hierarchy via the node refinement operation $NR(n_3, G_3)$. The edges e_m and e_l refer to the node n_3. Thus, they are critical for path-consistency.

Order of Operations and Parameters

Note that the order of operations is important here: one first has to add a node n_c in the coarse graph G_c before adding a new subordinate graph G_i (if G_c is empty, the hierarchy cannot be extended simply because there is no node in G_c to be refined). For practical purposes, one could provide a more convenient operation which adds a graph G_i without explicitly stating a node n_c: one could add G_i by specifying only the superordinate graph, G_c. The effect is that a new node n_c would be created in G_c as default behaviour. The operation $NR(n_c, G_i)$ can be performed subsequently. Given that n_c has no incident edges yet, path-consistency is not a problem in this case. Similarly, if the graph G_i is added without any parameters, it is assumed that the root graph G_r is its superordinate graph. So an according node n_c is created in the root graph which is refined to G_i.

UNDO A NODE REFINEMENT OPERATION. As a counterpart, we can **undo** a node refinement operation $NR(n_c,\ G_f)$ as well: First, the mapping $G_f := \rho_{G_r}(n_c)$ is removed from ρ_{G_r} and the edge (G_f, G_c) from $\mathcal{M}$. Then, for all edges e in all graphs (including those incident to n_c), the recursive border nodes starting with $\tau'_e(n_c)$ are all removed from τ'_e.

Recursive Application

However, this operation would result in an inconsistent meta graph $\mathcal{M}$ since G_f (together with its sub-tree) becomes loose. So we have to remove the problematic graph G_f *completely* from the hierarchy. Before G_f is actually removed, another measure must be taken: the undo operation is recursively applied on all nodes n_f in G_f which have subordinate graphs G_{ff} – the corresponding operations $NR(n_f, G_{ff})$ are undone, each. All subordinate graphs of G_f are then removed prior to G_f itself. One has to keep in mind that these intermediate operations are inconsistent. However, once the complete sub-tree rooted at G_f has been removed, $\mathcal{M}$ is again consistent. Edges with missing path linking functions have all been removed with the graphs. Consequently the composite operation as a whole is path-consistent.

Intuition

To clarify this idea, let us take a look at a concrete example: Supposed that we want to undo the node refinement operation $\mathsf{NR}(n_1, G_1)$ in Fig. 20. This would entail that $\mathsf{NR}(n_3, G_3)$ also has to be undone. The mappings $\tau'_{e_i}(n_1)$, $\tau'_{e_l}(n_1)$, $\tau'_{e_l}(n_3)$, and $\tau'_{e_m}(n_3)$ are therefore removed (note that $\tau'_{e_l}(n_2)$ stays unaffected, though). The graph G_3 is removed afterwards, so that the edge e_m becomes path-consistent again. In a final step, the graph G_1 is removed, so that the edges e_i and e_l become path-consistent once more.

REMOVE A GRAPH FROM THE HIERARCHY. This operation is already covered by the previous case of undoing a node refinement operation $\mathsf{NR}(n_c, G_i)$. It basically means the same in a system with a consistent meta graph $\mathcal{M}$.

REMOVE A NODE IN A GRAPH. This operation extends, in the most complicated case, the previous ones. First one has to find out whether the node n_i to be removed in G_i has a subordinate graph G_f. If this is true, the node refinement operation $\mathsf{NR}(n_i, G_f)$ is undone as laid out before. Otherwise this step is just skipped. The composite operation is consistent and path-consistent to this point. We can now remove all incident edges of n_i in the graph G_i (cf. the following operation) and subsequently n_i itself.

REMOVE AN EDGE IN A GRAPH. Suppose we want to remove the edge $e_i = (n_1, n_2)$ in a multigraph $G_i \in \Gamma$. The meta graph $\mathcal{M}$ stays intact anyway, so consistency is not an issue for this operation. If any of the edge's incident nodes n_1 and/or n_2 has a subordinate graph $G_{f1/2}$, the edge removal operation has to be carried out with caution to secure path-consistency: in this case we need to remove the appropriate (recursive) path linking functions $\tau^{(\prime)}_{e_i}$ entirely from $\mathcal{T}^{(\prime)}$. The edge e_i can be safely removed then.

Illustration

Consider an example for illustration: the edge e_k shall be removed from the hierarchical graph depicted in Fig. 20. All that needs to be done in the run-up to this operation is the removal of the (recursive) path linking function $\tau^{(\prime)}_{e_k}$ from $\mathcal{T}^{(\prime)}$. The same applies of course to the removal of any other edge, such as e_l or e_m.

ADD AN EDGE IN A GRAPH. Likewise, before adding a new edge $e_i = (n_a, n_b)$ to the multigraph $G_i \in \Gamma$, one has to check whether any node n_a, n_b incident to e_i has a subordinate graph. If this is the case, one has to specify along with the addition of the edge an appropriate set of recursive path linking operations $\mathsf{PL}'_k(e_i, n_k^{old}, n_k^{new})$. They start from $n_1^{old} := n_a/n_b$. The composite operation in turn works in an 'all-or-nothing' manner. Reasons for failure can be an incomplete set of path linking operations. Unchanged, the meta graph $\mathcal{M}$ remains consistent. Since correct recursive path linking operations yield a path-consistent state, the final state after the composite operation is applied is path-consistent.

4.2.6 *Derivation of Other Useful Operations*

Having defined the consistency of the basic operations, hierarchical graphs can be incrementally constructed in a correct manner. These basic operations are complete insofar as a hierarchical graph system can be realised with them. In spite of that, this set of operations may not always be convenient for creating or modifying hierarchical graphs.

Need for Supplementary Operations

To improve the usability, one could derive further composite operations which facilitate typical, recurring interactions with a hierarchical graph system. These additional operations can be seen as syntactic sugar. In the following passage, a preliminary list of useful operations is compiled. The list is non-exhaustive. In fact it can be extended with more operations if there is a specific requirement for these operations. The basic operations serve as building blocks for composing new operations, such as:

- merging two graph systems $\mathcal{G}_1 = (\Gamma_1, \rho_{G_{r1}}, \mathcal{T}'_1)$ and $\mathcal{G}_2 = (\Gamma_2, \rho_{G_{r2}}, \mathcal{T}'_2)$,
- connecting two nodes n_i and n_j which belong to different graphs G_i and G_j,
- defining a new subgraph G_{sub} in a graph G (for partitioning G).

Requirements

As it turned out, these operations are particularly useful in two respects: first, in a mediator-based architecture and second, for creating hierarchical graphs that represent indoor environments.

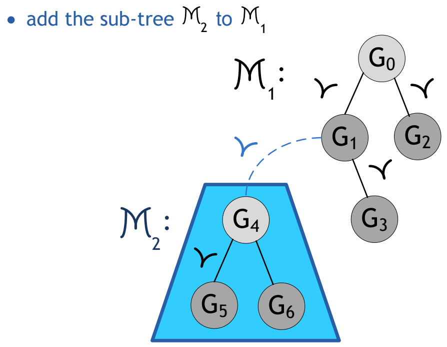


Figure 21: Merge Two Graph Systems

MERGE TWO GRAPH SYSTEMS. This operation is a generalisation of the 'add graph' operation. Instead of adding just one graph, we add a whole sub-tree $\mathcal{M}_2$ to $\mathcal{M}_1$.

Let $\mathcal{G}_1 = (\Gamma_1, \rho_{G_{r1}}, \mathcal{T}'_1)$ and $\mathcal{G}_2 = (\Gamma_2, \rho_{G_{r2}}, \mathcal{T}'_2)$ be two graph systems. The corresponding meta graphs are $\mathcal{M}_1$ and $\mathcal{M}_2$. The two graph systems can be merged into one graph system $\mathcal{G} = (\Gamma_1 \cup \Gamma_2, \rho_{G_{r1}} \cup \rho_{G_{r2}}, \mathcal{T}'_1 \cup \mathcal{T}'_2)$ by selecting a node n_1 in a graph $G_1 \in \Gamma_1$ for which we perform the node refinement operation $NR(n_1, G_{r2})$ (with the root G_{r2} of $\mathcal{M}_2$).

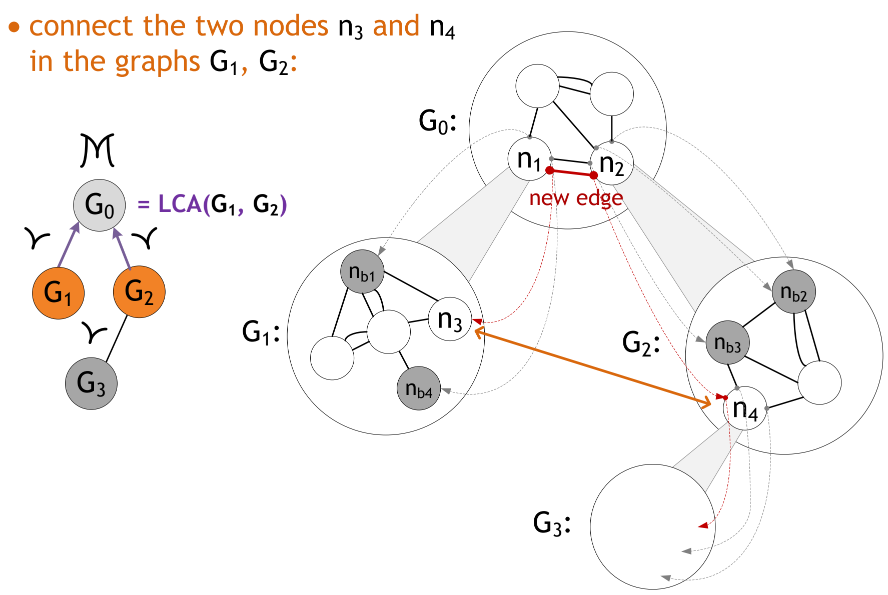


Figure 22: Connect Two Nodes in Different Graphs

CONNECT TWO NODES IN DIFFERENT GRAPHS. Instead of creating a new edge e in a higher-level graph together with a recursive path linking function τ'_e, we may want to directly specify lower-level nodes n_i and n_j, pertaining to different graphs G_i, G_j, which should be connected by that edge (see Fig. 22). This generalised operation obviates the need to specify a set of recursive path linking operations.[19] They can be directly derived from the hierarchy of graphs. The new operation is thus just a more comfortable way of specifying a new abstract edge e in a high-level graph.

Where the Edge is Created

Let n_i and n_j be two nodes of two different graphs $G_i, G_j \in \Gamma$. We want to connect them by an edge. This edge however has to be created in a graph further up the hierarchy. The two nodes appear instead somewhere in the recursive path linking function of this edge. Where the edge is actually created depends on the two nodes' position in the hierarchy: given that $\mathcal{M}$ is a tree, we can determine the graph G_{LCA} which is the least common ancestor of G_i and G_j in $\mathcal{M}$. Note that this could be one of the graphs themselves, if it occurs on the path of the other graph to the root. The new edge e is created in this graph G_{LCA}.

Defined Path Linking Functions

Concerning the new recursive path linking functions τ'_e, we go

[19] only some though, in case n_i or n_j are not leaves

down the path again from G_{LCA} to G_i and G_j. The graphs on the path are $G_k = \rho_{G_r}(n_k)$. The nodes n_k are superordinate nodes of these graphs. We start with n_1 in G_{LCA} by setting $\tau'_e(n_1) = n_2$ where $n_2 = \rho_{G_r}(n_1)$. We continue setting $\tau'_e(n_k) = n_{k+1}$ this way until n_{k+1} equals to n_i respectively n_j. Only if n_i or n_j have subordinate graphs, we need further assignments of recursive border nodes until leaf nodes are hit.

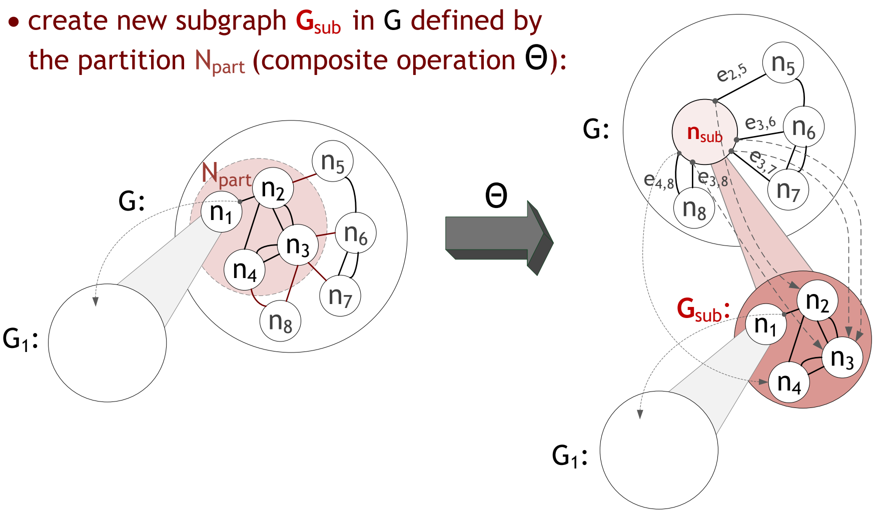

• create new subgraph G_sub in G defined by the partition N_part (composite operation Θ):
G:
N_part
n1
n2
n3
n4
n5
n6
n7
n8
G1:
Θ
G:
e2,5
n5
n_sub
e3,6
n6
e3,7
e4,8
e3,8
n7
n8
G_sub:
n1
n2
n3
n4
G1:

Figure 23: Partition a Graph by Creating a Subgraph

PARTITION SUBGRAPH. Supposed we have a graph G in a hierarchical graph system. Now we want to partition this graph internally by creating a new subgraph G_{sub} (see Fig. 23). This is the classic use for hierarchical graphs. For this purpose, one has to substitute in G the subgraph by a new node n_{sub}. That node is then refined to the subgraph G_{sub}. More precisely, the realisation of this composite operation looks as follows:

Simple Copy

The subgraph of $G = (N, E)$ is given by a node partition $N_{part} \subset N$. We create a new graph G_{sub} with the nodes of N_{sub} as node set by copying all nodes in the partition to G_{sub} (not a deep copy, though, since subordinate graphs etc. are not considered at first). To obtain a consistent meta graph $\mathcal{M}$, the new graph G_{sub} is linked to the hierarchy by creating a new node n_{sub} in G and a subsequent node refinement operation $NR(n_{sub}, G_{sub})$. It establishes the connection (G_{sub}, G) in $\mathcal{M}$. For all edges $e = (n_1, n_2) \in E$ with $n_1, n_2 \in N_{part}$ (i.e. internal edges of the subgraph), we likewise create corresponding edges $e_{copy} = (n_{1copy}, n_{2copy})$ in G_{sub} (not a deep copy either). So far, these operations are all path-consistent.

The goal now is to replace the nodes and edges of G in the partition N_{part} by the new graph G_{sub}, respectively n_{sub} as superordinate node. Two things need to be done to achieve this goal:

Correcting Nodes

First we have to redirect for all $n \in N_{part}$ any mappings of the form $\rho_{G_r}(n) = G_f$ to $\rho_{G_r}(n_{copy}) = G_f$. Thus the corresponding hierarchy relation edges (G_f, G) in $\mathcal{M}$ have to be redirected to (G_f, G_{sub}) as well. In general, this intermediate step is consistent, but not path-consistent for ρ_{G_r} and $\mathcal{T}'$ do not correspond anymore. By changing the node refinement function, we also need to readjust the corresponding (recursive) path linking functions. In Fig. 23 for instance, the edge $e_{1,2} = (n_1, n_2)$ becomes temporarily inconsistent after linking G_1 to G_{sub}.

Correcting Edges

This path-inconsistency is corrected in the second step: All edges $e = (n_1, n_2) \in E$ with $n_1 \in N_{part}$ and $n_2 \in N \setminus N_{part}$ are redirected to $e = (n_{sub}, n_2)$ with a simultaneous adjustment of the (recursive) path linking function: the mapping $\tau_e^{(\prime)}(n_1) = n_b$ becomes $\tau_e^{(\prime)}(n_{sub}) = n_{1copy}$ and $\tau_e^{(\prime)}(n_{1copy}) = n_b$. The same change of the path linking function of course applies to all other edges e in higher-level graphs $G_c \neq G$ with $\tau_e^{(\prime)}(n_1) = n_b$ as well. For these edges, the node n_1 in G plays the role of a border node. The path linking functions are adjusted in all these cases.

Moving Path Linking Functions for Internal Edges

Additionally, the mappings in the (recursive) path linking functions $\tau_e^{(\prime)}$ for all internal edges $e = (n_1, n_2) \in E$ with $n_1, n_2 \in N_{part}$ are all adopted in new (recursive) path linking functions $\tau_{e_{copy}}^{(\prime)}$. The only modification is that appearances of n_1 and n_2 are replaced by n_{1copy} and n_{2copy}, respectively. After this, the original (recursive) path linking functions $\tau_e^{(\prime)}$ can all be removed from $\mathcal{T}^{(\prime)}$. Such an example is the internal edge $e_{1,2} = (n_1, n_2)$ of the graph G in Fig. 23: the function $\tau_{e_{1,2}}^{(\prime)}$ is affected. Its entire contents are copied to $\tau_{e_{1,2copy}}^{(\prime)}$ before it is deleted. So $\tau_{e_{1,2copy}}^{(\prime)}(n_{1copy})$ in the new function corresponds to the former value $\tau_{e_{1,2}}^{(\prime)}(n_1)$ of the old function.

Now the nodes and induced edges of the partition N_{part} have been completely replaced by their copies in G_{sub}, with incident edges and node refinements all redirected. The nodes and induced edges of the partition N_{part} are now isolated in G. All that remains to be done is to remove them from G.

Merging Two Graphs via the Operation

Creating a subgraph is a very useful operation. Note that it can for example be used to merge two different graphs G_i and G_j which are siblings (i.e. have a common superordinate graph G_c): The corresponding nodes $n_{i/j}$ in the graph G_c with $\rho_{G_r}(n_{i/j}) = G_{i/j}$ are put together. They form a new subgraph in G_c.

4.3 SUMMARY

This chapter introduced a system architecture based on the central concept of a hierarchical graph. In particular, it has been proposed for the integration of distributed spatial networks. The first section motivated the need for such a system. To handle the information flow between distributed elements in the system, the notion of a mediator has been developed. The representation of context information has been discussed together with the resulting constraints that affect path finding. Three different options have been explored for incorporating context information into the path finding process, including an analysis of their respective strengths and weaknesses. Two of them have been realised in an initial implementation of the system architecture.

Besides the practical aspects of the system, this chapter also explained the theoretical issues of a hierarchical graph system. Some basic concepts like meta graph or hierarchical relation have been formally defined together with a set of basic operations (e.g. node refinement). These operations serve as primitives in the incremental construction of a hierarchical graph. The consistency of this construction process has been investigated in particular.

Now that the general design issues of a hierarchical graph system have been covered, we can focus our attention on the concrete modelling of indoor environments. The following chapter explains how these environments fit into the proposed hierarchical graph model.

5 APPLYING HIERARCHICAL GRAPHS TO INDOOR ENVIRONMENTS

In this chapter a closer look is taken at the specific application domain of indoor environments. The focus is on pedestrian navigation in buildings because these are the most interesting environments from a research point of view. Unlike outdoor networks, they have not been treated in a systematic way. There is no standard model for navigation in indoor environments (see Chap. 2).

Two aspects are mainly investigated: firstly, how a graph hierarchy can be automatically created for indoor environments and secondly, how this hierarchy can be exploited to guide path finding and produce human-oriented route descriptions. A pragmatic method is described to obtain a hierarchical graph model of a building. The method starts from a given set of floor plans (see Section 5.3). The generated model serves as a basis for a human-centered navigation assistance system.

In Section 5.1 the general benefits of a hierarchical representation are briefly explained. Section 5.2 gives an overview of what a complete system architecture looks like, which is more wide-ranging and goes beyond the prototypical implementation achieved in this work. The next section (5.3) looks at the details of the transformation of a geometry-based representation into a multi-level, versatile graph structure. Further aspects and arguments for extending this initial hierarchy are discussed in Sect. 5.4. There, two new algorithms are proposed which are motivated by practical navigation problems.

For generating meaningful route descriptions in particular, it is useful to maintain some parts of the geometry so that spatial relations can be derived. Featuring *both* geometry and graph structure, the proposed model is *hybrid*. The benefits of using such a hybrid model are discussed and illustrated by concrete examples. In Section 5.5 the hierarchical model is applied to concrete navigation problems in buildings. Section refsec:scenarios gives an outlook by presenting three possible scenarios for using such a system. The chapter ends with a conclusion in Sect. 5.7.

5.1 WHY HIERARCHICAL GRAPHS FOR NAVIGATION IN BUILDINGS?

First of all one may well ask why it makes sense to apply hierarchical graphs for navigation in buildings. There are several arguments in favour of hierarchical graphs:

Speed-Up Necessary?

The first impression is that efficiency plays rather a minor role because the graphs are relatively small compared to large traffic networks outdoors. However, for 'lively' environments such as airports, museums, etc., this is not necessarily the case: they are frequented by many people so that many queries are likely to be posed within short amounts of time. Efficient data organisation and query processing therefore become significant questions for navigation tasks in these environments. Building up a hierarchy does not pay off so much for just one query, but for many resembling queries (whenever recomputations of the same partial results can be avoided). The hierarchy can be seen as an index structure; as such it has to be computed only once.

Richer Model

Moreover, hierarchical graphs provide a richer model: In fact, many spatial planning problems can be thought of *hierarchically*. Human indoor navigation is an excellent example for demonstrating the virtues of applying hierarchical planning. Consequently navigation tasks can be carried out at several levels of detail, from a coarse, high-level plan (determining the main course, i.e. which floors and stairways to take) to detailed plans (determining local choices, e.g. between several doorways or for the circumvention of obstacles). The resulting advantages are that path finding and the subsequent description of paths can be structured in a way which strongly resembles the way people solve spatial problems (i.e. to provide 'meaningful' route descriptions).

Integration with other Networks

Last but not least, a hierarchical graph model alleviates the integration with other spatial networks into a comprehensive geospatial world model. An architecture like the mediator-based architecture proposed in the previous chapter can be employed for this purpose.

Exemplary Route Descriptions

To be more precise, let us consider some typical examples for indoor route descriptions given by people: If you are at the first floor of a large building, and you ask someone how to get to a particular room, the explanation may well start with *"Go to the third floor ..."*.

What is essentially behind this is a two-level (or, in general, multilevel) hierarchical model of the building. An example is depicted in Fig. 24.

Hierarchical Planning

The upper hierarchy level consists of all floors, while the lower level models the topology of each floor. In addition, the hierarchy shown in Fig. 24 also has an intermediate level defined by wings. Navigation between different floors usually consists of the steps *"go to the lift (staircase etc.)"*, *"go to the target storey"*, and *"navigate the target storey"*. This is a typical case of hierarchical planning as it has been inves-

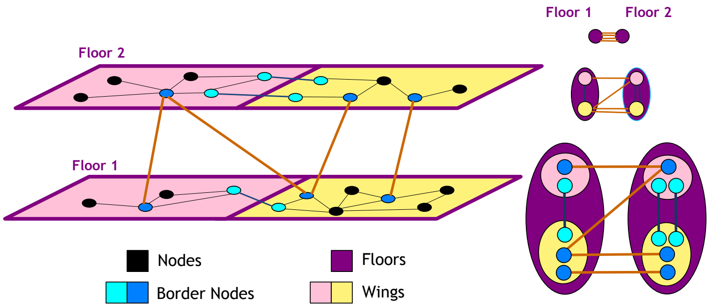


Figure 24: Hierarchical Graph of a Building

tigated in Artificial Intelligence for decades [Sac73]. Therefore the intention of our model is to support hierarchical planning, namely by providing hierarchical graphs.

Structuring Indoor Spaces

The primary use for a graph hierarchy is of course the representation of different floor levels in a building. Other use cases may necessitate a more detailed representation comprising different wings or sections of a building (as in Fig. 24). Together, floors, wings, and their constituent rooms and corridors yield a hierarchy of four levels (if the building as a whole is considered as the root). Depending on the intended application, it could make sense to subdivide some rooms or corridors further, e.g. to better cope with their complex structure. The resulting hierarchy would then have five or more levels. One can profit from this richer structure too for giving route descriptions at several levels of detail [DGP03, Tom07]. If necessary a wayfinder may refine an abstract description and see its individual steps. This kind of interaction is very convenient compared to flat, static instructions which are standard.

Integration with Other Networks

However, extensions should also be considered in the other direction: there may also be further levels above the level of floors. If we want to represent not only a single building, but the whole campus of a university (or another similar complex consisting of many buildings) for instance, or buildings scattered over the city, each building would be represented as a node in the graph one level above the level of floors. The edges in the graph at the 'building level' represent walkways or streets. More generally, this aspect concerns the **integration** of separate graph models of a certain detail level into a graph model with a higher level of abstraction.

Containment

A further use of hierarchical graphs can be the representation of areas which are contained within each other. As an example, think of the vegetable section in a hypermarket. The vegetable section may be subdivided into areas with salad, cucumbers, carrots, etc. In the hierarchical graph model we would have a node for the entire vegetable section at some level n, and this node refers to the graph of the salad, cucumber etc. areas at level $n - 1$.

Maps at Different Detail Levels

The hierarchy may also reflect the underlying data organisation.

For example, one could have a library modelled as one room in a floor plan. In a second floor plan however, the internal details of the library may be modelled. We could link these two representations via node refinement in a hierarchical graph. Since the library is a special part of a building, this separation may be useful knowledge. For path finding e.g., one would explore the internals of the library only if necessary. The whole library could also be marked as 'private' or 'staff only' – this is an elegant way for excluding it from general path search.

5.2 OVERVIEW

This chapter explains the basic principles and methods necessary for an indoor navigation system based on a hierarchy of graphs. Before going into the details for solving individual subtasks, let us give an impression of what a *complete* system could actually look like. In Fig. 25, a possible system design is sketched:

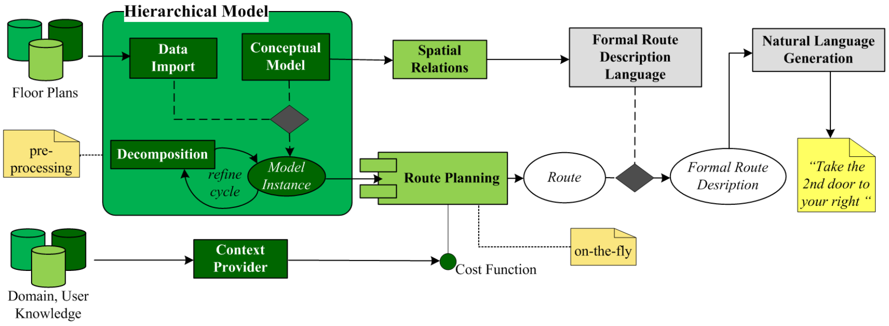


Figure 25: Indoor Navigation: System Design with Components

Current Status

Note that the complete architecture shown in Fig. 25 is more wide-ranging than the work that has actually been carried out in the course of this thesis: not all components have been implemented due to the limited time for this research. Nonetheless, the above diagram is well suited to give a big picture, illustrating the functionality and interdependecies of individual components. As such, it can be regarded as guideline for future implementations of the remaining components (see Chap. 6 and 8 for a comprehensive overview). In the present work, we concentrate on the components marked in green. They have also been implemented.

Components

On the left hand side, we have the basic data and input for the system. The results are shown on the right hand side. Floor plans form the basis, together with additional domain and user knowledge which can be linked to the floor plans (such as Points of Interest etc.). The output of the system is twofold: we have routes which represent paths along the graph model *and* descriptions of these paths. So one can distinguish between a component for providing paths and one for providing their descriptions. Therefore, a modular architecture is proposed. The hierarchical graph model is of course the central element in this architecture. It serves as a reference for both components. The general model has been discussed in Sect. 4.

The emerging question is: how can a hierarchical model be constructed from the original data? This task is carried out in the data importer and decomposition modules. We therefore start with the details of these components.

5.3 MODELLING ASPECTS AND CONSTRUCTION OF THE HIERARCHY

5.3.1 *Interpretation of Floor Plans*

Addressed Questions

The first step towards a comprehensive, automated data processing method concerns data acquisition. This is a practical but nonetheless important issue because the basic data format is the starting point for all further considerations: *How is an indoor environment represented, and how can advanced navigation tasks be carried out based on this representation?* (i.e., which spatial relations are available or can be derived from it?) It may be necessary to enrich the basic structure. *What kind of information is required for this purpose and how can a graph structure be – automatically – constructed?*

Indoor vs. Outdoor

Data acquisition for indoor environments is unfortunately more problematic than modelling, for example, outdoor spatial networks. The crucial difference lies in the type of data available: Aerial images (e.g. provided by satellites) facilitate the automatic detection of an entire transportation network (such as a highway system). The modelled road segments and intersections already represent the desired navigational structure. By contrast, the primary source of data for indoor environments is a collection of floor plans. Therefore they are the starting point for our considerations.[1]

Floor Plans are Complex

The problem with floor plans is that they are centered on a building's geometry. Thus one can *not* directly make use of a graph structure for giving navigational advice. From the original floor plan data, such a graph has to be derived first. As discussed in Sect. 2.1 there is not one unique representation for the interior of a building, but instead several different kinds of graph structures can be used for navigation. It is arguable which one to choose, although a minimum set of assumptions on the walking behavior of people would be helpful.

Conceptually, floor plans can be seen from two different perspectives:

ARCHITECT'S PERSPECTIVE. From the point of view of an architect, floor plans document the structural aspects and the shape of

[1] Alternative approaches are also thinkable, especially in the field of robot navigation: Autonomous robots can, e.g., acquire knowledge of their environment via sensors. For this purpose, 180° or, more recently, 360° laser scanners are employed. However, the geometry reflects the structure of the floor plan, so this is just another method for *obtaining* floor plan data (without prior knowledge of it).

a building. They are technical drawings (blueprints), mainly serving as an aid for construction. Additionally, they can be enriched by diverse architectural symbols and measurements. This makes automatic image processing particularly difficult. We therefore concentrate on 'pure' floor plan data without any symbols or measurements. It is assumed that a vector-based representation is available. This representation can, e.g. originate from a CAD system or a manual drawing with another similar tool.

WAYFINDER'S PERSPECTIVE. Floor plans are, on the other hand, frequently used as *maps* of buildings for both visitors and occupants. Examples include emergency- and evacuation plans or you-are-here (YAH) maps. They are shown on strategic points of a building to aid human wayfinders. Since we address *human* indoor navigation, our focus is on this second aspect – floor plans are primarily considered as maps. So we have to move from the purely geometry-based model towards a qualitative form of representation more akin to people's understanding of a map. In order to get to such a map representation we have to *interpret* floor plans. For a computer system however, this implies making sense of the low-level geometric concepts and deriving high-level navigational knowledge from them (similar to the way humans look at maps and reason).

We propose a systematic method for interpreting floor plans for navigation. It consists of the following individual steps:

- scan floor plans on paper (preliminary step),
- vectorise each floor plan (e.g. with a CAD system or similar tool),
- extract a flat, basic graph structure from each floor plan for navigation,
- connect different floor plans and create a hierarchical graph.

For our further considerations it is assumed that a set of floor plans of the environment is available in a vector-based format (these data originate e.g. from a CAD application). Since non-proprietary CAD systems are scarce to find, we decided to use a similar system devised for modelling buildings. It is called Yamamoto [SH06]: Initially started as a student project, the system has been developed further as an open source project at the university of Saarbrücken.

Nonetheless, it is sensible to represent the basic elements of a floor plan in a very simple, general form (independent of a particular CAD system/tool). For the systematic method we hence need to address the issues of:

1. finding an appropriate geometric representation
2. deriving an interpretation of the included geometric configurations in terms of a flat graph.

In the following sections, we adhere to this simple method and explain the basic steps for converting the geometric representation naturally occurring in floor plans to a basic graph structure. In Sect. 5.4, we then describe our experiences with this method and the resulting graph model when it is actually applied to real floor plans.

5.3.2 *The Underlying Geometric Model*

This section presents the underlying geometric model from a vectorised floor plan. Before we focus our attention on the resulting navigation problems, this basic data model is explained. The geometric model serves as a reference for the hierarchical model, and at the same time as an ontology for spatial elements of a building. To define an ontology for wayfinding in buildings one has to understand the *structure* of buildings first, including the resulting free space where people can move and of course boundaries and other obstacles.

5.3.2.1 *Planar Floor Plan and Dual Regions*

We start with a vectorised, yet very primitive floor plan in the beginning – this model consists *only* of boundary lines and their respective end points. Even the basic spatial regions (enclosed areas) are left implicit. In other words, we assume that only a **planar graph** is available. This abstraction is convenient because it allows us to consider floor plans in a formal way, without any preference for a particular system or format:

Definition 5.3.1 (Planar Floor Plan Graph). A **planar floor plan graph** $G_f = (N_f, E_f := E_b \cup E_o)$ is an undirected graph with a planar embedding in a two-dimensional spatial reference system.[2] We differentiate between two types of edges: the first edge type, E_b, represents a straight-line boundary segment. The second type, E_o, represents an **opening** between two such boundary segments.[3] Nodes $n = (x, y) \in N_f$ demarcate the end points of both edge types. Vice versa, the edges $e = (n_a, n_b) \in E_f$ represent the lines $\overline{n_a n_b}$.

For planar graphs the following properties are known [Die00]:

- no two line segments cross each other, i.e. edges intersect only at their end points[4]
- there are regions implicitly bounded by edges (in literature, they are termed 'faces'), including the infinite exterior face. An edge generally separates two faces.

Our intention is to derive the implicit regions from the floor plan graph and use them for further processing. They should be ideally represented in a standard way, so that manipulation with both simple and advanced algorithms is facilitated. Polygons (in the sense of

[2] Note that the reference system does not to need be global (such as WGS-84). It can be local, i.e. adopt the dimensions of the scanned drawing.

[3] This can be, for example, a door (see Fig. 26)

[4] a reasonable assumption if the floor plan has been vectorised properly

Computational Geometry; see also Def. 5.3.3 in Sect. 5.3.2.2) are such a suitable representation.

Since planar floor plan graphs (with their included regions) are very abstract, one may need to extend this formalism with more concrete concepts. If for example a navigation application requires distinguishing between different types of regions (say between ramps and stairs for persons in a wheelchair), one should be able to annotate these regions with the respective types/concepts. Our intention is to provide a basic model which is extensible in this respect.

The distinction between two edge types is drawn for practical reasons: whereas both delimit the boundaries of a region, only the second type, E_o, represents (in an abstract manner) any connection on the boundary that allows us to move from one region to another. In the literature these special elements are also referred to as *gateways* [RT05] or *exits* [HL04]. The other type, E_b, represents hard/impenetrable boundaries. Examples include not only physical boundaries like walls, but any other form of division like height difference (think of loges in a theatre, or platforms in a train station).

It is even foreseen that entries to ramps, staircases or elevators are modelled as openings. Staircases and elevators are special regions (they have to be explicitly annotated as such). They play a major role in connecting different floor plans. In Sect. 5.3.4, this special case is discussed in more detail.

With the two different edge types, we already have a small ontology that carries the basic semantics of navigation. To include more sophisticated constraints like opening hours or access restrictions, it makes sense to allow additional annotations for edges in E_o. This topic is further discussed in Sect. 5.5. However, the basic categorisation into E_o tells us which edges on the boundary are generally traversable for a human wayfinder.

To get an impression of concrete models with planar floor plan graphs, consider the two examples shown in Fig. 26:

Both floor plan graphs are valid representations of the same architectural elements, albeit on different scales. The essential difference is that in the first graph walls have a certain thickness, whereas in the second graph this thickness is idealised to a line. So walls and doors are either modelled as proper regions with boundaries or as boundary segments alone (depending on the choice of scale). Instead of stipulating a certain way of modelling, there is more flexibility if both models are allowed. We therefore want to take into account *both* ways of modelling.

The corresponding enclosed regions of the previous example, except for the outer face, are depicted in Fig. 27. Notice that walls and doors are additional regions in the example with the finer scale.

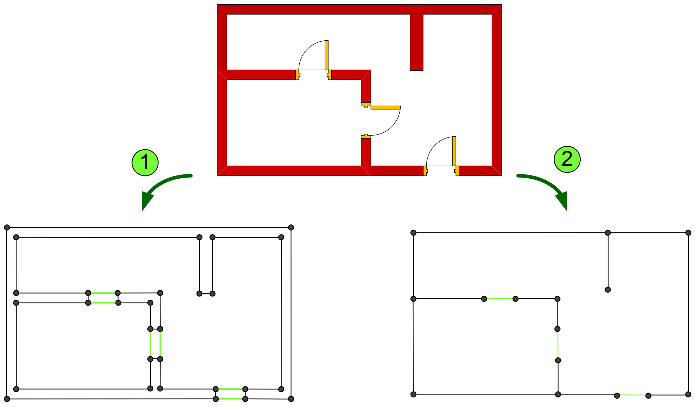


Figure 26: Example for Two Planar Floor Plan Graphs

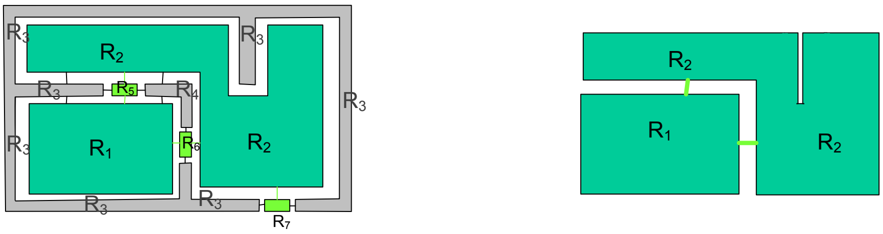


Figure 27: Implicit Regions Enclosed by Edges

How can we characterise these regions more specifically (e.g. to keep walls and ordinary rooms apart)? One just has to take a look at the included boundary types for this purpose: if there are only E_b boundary segments (no openings), the regions can neither be entered nor left. Generally speaking these regions qualify as obstacles. Walls with a certain thickness or other non-enterable objects fall into this category. The other kind of regions have some walkable connections of type E_o to the outside or to neighbouring regions (in the sense of navigation). These regions could be rooms, stairs, doors or ramps.

Now we are interested in a method to determine these implicit faces and their relations from a given planar floor plan graph, that is deriving the so-called **dual graph**.[5] Resembling examples for this sort of graphs are Region Adjacency Graphs [LOS06, PG96], although there is usually only one type of edges.

For our purposes it makes sense to extend this notion to two edge types: from a **dual** perspective, one or more shared edges of type E_b are considered as *one* neighbouring relation (*external connection*) between two faces/regions. In contrast, *each* edge of type E_o is considered as one transition relation (*path connection*) between two faces/regions (meaning that there can be, in general, multiple path connections between two regions). The resulting graph structure is

[5] the nodes in this graph are regions and edges represent the existence of connections between regions.

presented in Sect. 5.3.3, where we go into more details regarding these relations.

Notably, the faces in a floor plan graph may have both *outer* and *inner* boundaries (e.g. when other regions are contained in them). An example is shown in Fig. 28:

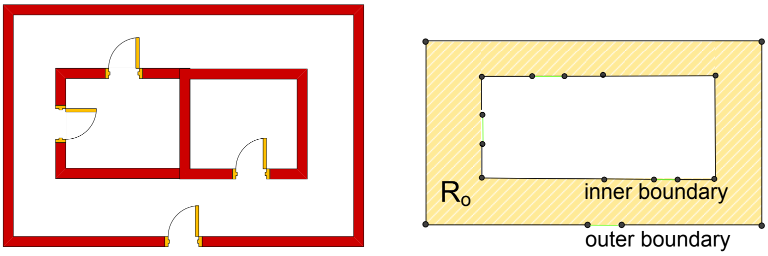


Figure 28: Circular Corridor R_0 with Inner *and* Outer Boundary

This case can occasionally be found in real floor plans, for example of airports (where there are individual duty-free shops in the departure or waiting area), universities (with circular hallways around some lecture rooms) or other large buildings with similar features. In the example of Fig. 28, the common outer boundary of two regions constitutes the inner boundary of the containing region. So when determining the boundaries of a region one also has to look out for a possible nesting. In general, the nesting can be recursive.

To identify this problematic constellation, it would be ideal to have the outer boundaries of regions represented as polygons. One could then use a standard algorithm from Computational Geometry, e.g. to detect whether points of one polygon lie in the interior of another polygon (with a simple ray crossing method).

5.3.2.2 *Deriving Regions from the Floor Plan Graph*

As basic topological relations we only have cases in which

1. regions share boundaries (E_b with or without E_o),
2. regions are nested in one another (i.e. share some *inner* boundary),
3. regions are disconnected (none of the above relations holds).[6]

All other topological relations [EF91] are not relevant for our considerations of floor plan graphs.[7]

Consequently we can see the major parts of a floor plan graph (with the relations of point 1) in an intuitive way as a connected **mesh of polygons**:

[6] overlapping is not possible because of the planarity condition (i.e. all edges intersect at their end points only).

[7] for example, tangential containment is not considered here because the larger region would not be minimal.

Definition 5.3.2 (**Mesh** in a Floor Plan Graph.)**.** A **mesh** in the context of a floor plan graph G_f is a connected component G_m of the graph G_f, i.e. all nodes and edges which can be reached from a given start node $m \in N_f$. All regions bounded exclusively by edges in G_m are called **regions in the mesh** G_m (see the example in Fig. 29).

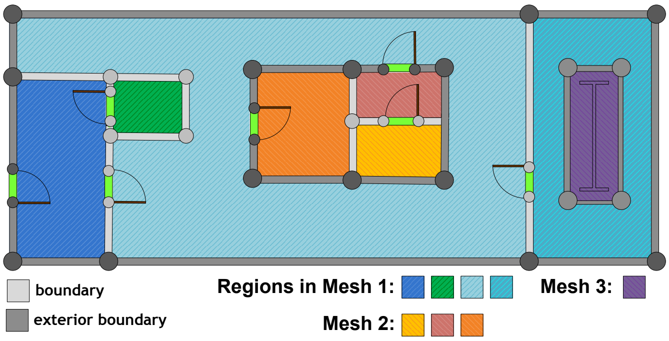


Figure 29: Exemplary Floor Plan with Regions in Different Meshes

Definition 5.3.3 (**Polygon** (Computational Geometry))**.** A **polygon** $P := C_1, C_2, .., C_{z \geqslant 3}$ in the sense of Computational Geometry is a sequential list of *distinct* corners $C_{i \in 1..z} = (x_i, y_i)$[8] where neighbouring corners are connected via boundary lines. Moreover, the tail C_z is linked to the head C_1. All corners are ordered counter-clockwise (ccw). Two conditions are crucial for polygons:

1. no corner C_i may appear repeatedly in the list (i.e. two boundary lines may not touch each other, except if they are neighboured),
2. no pair of boundary lines $\overline{C_iC_{i+1}}$ and $\overline{C_jC_{j+1}}$ (with $i \neq j$) may cross each other, i.e. there may not be any intersection points $C_x := \overline{C_iC_{i+1}} \cap \overline{C_jC_{j+1}}$ outside the list of corners.

Polygons correspond to elementary cycles in the floor plan graph. These are all minimum cycles which do not contain any other cycle. The principal idea is to traverse the planar floor plan with a simple graph search algorithm and determine these minimum cycles. The algorithm can proceed in a mesh from one region to another via their shared boundary (an edge in the floor plan graph). To determine *all* regions, one just needs to make sure that every edge is visited exactly *twice* (in possibly opposite directions), for two different cycles/regions.

Despite the fact that edges in the floor plan graph are *undirected*, a traversal by a graph algorithm does naturally impose an order on the edges. We therefore distinguish between two conceptually different notions:

- an *undirected* edge $e_f = \overline{AB}$ in a floor plan graph,

[8] indices are calculated **modulo** z

- two *directed* traversals (visits) of the edge, represented by the vectors $\vec{e_1} = \overrightarrow{AB}$, $\vec{e_2} = \overrightarrow{BA}$ (they correspond to the *ordered* list of points A, B respectively B, A).

We refer to the latter also as **directed edge**, in contrast to the undirected counterpart in a floor plan graph. An apt concatenation of these directed edges should result in the polygons which correspond to the minimal regions.

BASIC ALGORITHM. In planar floor plan graphs regions are not *explicitly* represented. Nonetheless, a surrounding set of edges positively defines a region. An algorithm is therefore needed which automatically determines all regions enclosed in a floor plan. Polygons are an ideal representation for these regions because they are the basic data structure used in Computational Geometry. The purpose of the following algorithm is to obtain a set of polygons for a given floor plan graph:

Generally, when we start going through the floor plan graph from any directed edge $\overrightarrow{AB}$, there are several adjoining edges $\overline{BC_1}, .., \overline{BC_n} \in E_f$ (with $C_i \neq A$) where we can continue. We have to decide which of these edges is the next in the counter-clockwise order of the currently constructed polygon.[9] The special case where no consecutive edge $\overline{BC_i}$ exists (i.e. node B being a dead end in the floor plan graph) can of course also occur. This problem is deferred for the moment, since it complicates understanding the basic idea of the algorithm. It is covered along with other special cases in the next section.

The strategy for choosing the next directed edge $\overrightarrow{BC_i}$ along the polygon's boundary is a core component in the processing of a floor plan graph, yet very simple. It is sketched in Fig. 30:

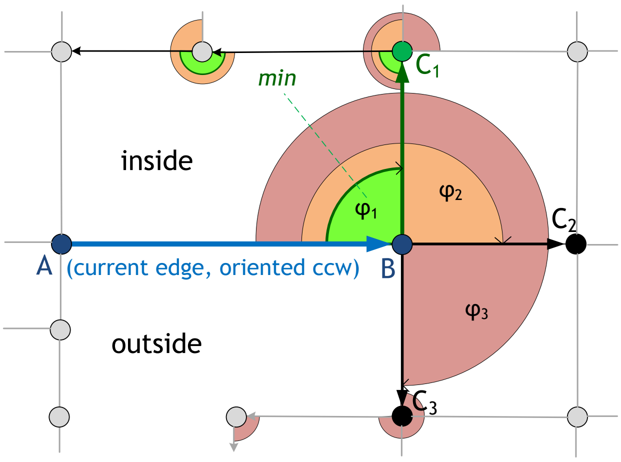


Figure 30: Strategy for Choosing the Next Boundary Line

Assuming we are currently at the directed edge $\overrightarrow{AB}$, the point C_i on the next directed edge $\overrightarrow{BC_i}$ is part of the polygon iff we choose $\overline{BC_i}$

[9] the order of the polygon is only counter-clockwise if the first edge is also oriented in this direction. However, this condition can be ensured quite easily (see below).

such that the angle $\varphi_i = \measuredangle(ABC_i)$ (enclosed between $\overrightarrow{AB}$ and $\overrightarrow{BC_i}$) is minimal. In Fig. 30, the minimum angle is depicted by the green sectors.

The intuition is to find the leftmost point C_i as defined in Computational Geometry (the *interior* of a polygon is normally to the left of a counter-clockwise oriented edge $\overrightarrow{BC_i}$ on the polygon's boundary). By selecting the minimum angle we ensure that the respective edge $\overline{BC_i}$ forms a minimum cycle in G_f in the sense that no other region is enclosed: if we took another point C_j instead, we would not obtain the *minimum* region (in some cases not even a polygon at all) because the edge $\overline{BC_i}$ would be enclosed in it. However, by definition an edge separates exactly two regions.

Instead of using trigonometric functions – which are quite expensive in terms of computation – there is a much simpler and elegant solution without actually calculating the angle. We can resort to means of Computational Geometry (see Alg. 1):

```
Algorithm 1: Find Next Directed Edge
Input: A directed edge $\overrightarrow{AB}$ in an underlying floor plan graph $G_f$.
Output: The leftmost directed edge $\overrightarrow{BC_i}$ with the minimum angle
        $\varphi_i = \measuredangle(ABC_i)$.
let $C_1, .., C_n$ denote the list of nodes adjacent to $B$ (without $A$) ;
choose the first point $C_1$ as leftmost point $C_{min}$ ;
for j = 2 to n do
    test whether the next point $C_j$ is located to the left of $\overrightarrow{BC_{min}}$,
    i.e. whether
    $(C_{min}.x - B.x)(C_j.y - B.y) - (C_{min}.y - B.y)(C_j.x - B.x) > 0$;
    if $C_j$ is to the left of $\overrightarrow{BC_{min}}$ then $C_j$ becomes the new leftmost
    point $C_{min}$ (otherwise it remains $C_i$) ;
end
return $\overrightarrow{BC_{min}}$
```

One can go through the entire list of nodes $C_1, .., C_n$ and compare each element with C_{min}, the best candidate so far. The next edge with the minimum angle can be determined efficiently this way, in linear time (w.r.t. the list of adjacent nodes). If we choose, in place of the minimum angle, the edge with the *maximum* angle (illustrated by the red sectors in Fig. 30) and start from an edge on the exterior boundary of a mesh, we obtain the largest, outermost polygon that encloses all other edges and regions in the mesh. This is particularly useful for determining the nesting between different meshes. Consequently we can generalise the traversal through a floor plan graph and provide a convenient method for constructing minimum and maximum polygons (see Alg. 2):

In order to detect the outer boundary of a mesh, we need an appropriate starting parameter for the directed edge $\overrightarrow{AB}$ which is already on this boundary. One could for example find this edge by con-

Algorithm 2: Construct the Minimal/Maximal Polygon

```
Input: A directed edge $\overrightarrow{AB}$ in an underlying floor plan graph $G_f$,
       an optimisation parameter $minOrMax$.
Output: A polygon $P_{A,B}$ starting with $A, B$ and describing the
        minimum/maximum region from it.
1 add $A, B$ to $P_{A,B}$ ;
2 $nextEdge = \overrightarrow{AB}$ ;
3 repeat
4     $nextEdge$ := find next directed edge $\overrightarrow{C_b C_c}$ from $nextEdge$
      with $minOrMax$ angle ;
5     if $nextEdge \mathrel{!=} \overrightarrow{AB}$ then add $C_c$ to $P_{A,B}$ ;
6 until $nextEdge = \overrightarrow{AB}$ ;
7 return $P_{A,B}$ ;
```

necting the lowest rightmost node n_{lr} in the mesh with the leftmost node n_{al} adjacent to it. Note that this has to be done in reverse order (i.e. $B := n_{lr}$ and $A := n_{al}$) for obtaining counter-clockwise order.

Finally, we can put together the individual pieces into a comprehensive algorithm (see Alg. 3 below). It constructs polygons for all implicit regions in a given floor plan graph, plus for the exterior boundaries of all meshes:
Let us take the previous floor plan of Fig. 29 as a concrete example. Fig. 31 illustrates the individual steps of the algorithm on this floor plan graph. The sequence of steps can be directly read from their numbering.

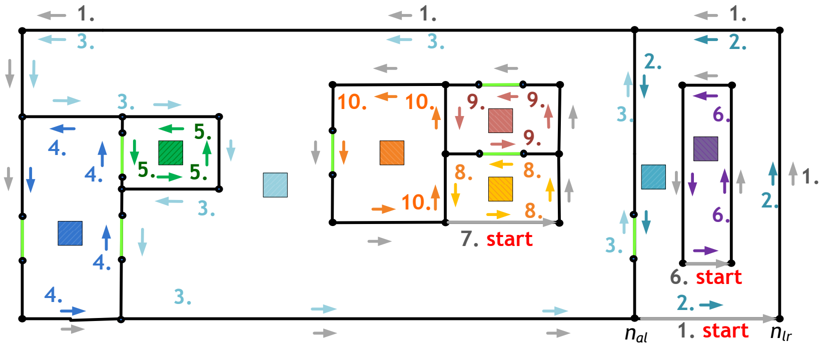


Figure 31: Algorithm for Finding Polygons in a Floor Plan Graph

The way the algorithm works can be explained intuitively: Every undirected edge e_f in a floor plan graph has a counter $visited(e_f)$. It keeps track of the number of times e_f has been traversed by the algorithm so far. Accordingly, there is $lastVisit(e_f)$ as corresponding directed edge of the last traversal.

The algorithm starts with the extreme (outermost) directed edge $start$ on the exterior boundary of a mesh (*line* 3) and finds this com-

[10] the operation *flip* on a directed edge $\overrightarrow{AB}$ simply yields its reverse edge $\overrightarrow{BA}$.

Algorithm 3: Find Regions in a Floor Plan Graph

```
   Input: A planar floor plan graph $G_f = (N_f, E_f)$.
   Output: A set $\mathcal{R}$ of polygons, one for every region enclosed in $G_f$
           and a set $\mathcal{O}$ of polygons describing the exterior
           boundaries of all meshes in $G_f$.
1  Let $\mathcal{R}$ and $\mathcal{O}$ be empty and $\mathtt{visited}(e_f) = 0$ for all $e_f$ in $E_f$ ;
2  while there is an edge $e \in E_f$ with $\mathtt{visited}(e) = 0$ do
3      start := lowest, rightmost directed edge $\overrightarrow{n_{al}n_{lr}}$ of all
       non-visited edges ;
4      borderOfMesh := constructPolygon(start, MAX) ;
5      add borderOfMesh to $\mathcal{O}$ ;
6      foreach undirected edge $e_f$ corresponding to a directed edge $\overrightarrow{e_m}$ in
       borderOfMesh do lastVisit($e_f$) = $\overrightarrow{e_m}$, visited($e_f$)++ ;
7      while there is an undirected edge $e_f$ in $E_f$ with visited($e_f$) = 1 do
8          if $e_f$ appears in borderOfMesh then $\overrightarrow{e_m}$ = lastVisit($e_f$)
           else $\overrightarrow{e_m}$ = flip(lastVisit($e_f$)) ;
           10
9
10         Poly := constructPolygon($\overrightarrow{e_m}$, MIN) ;
11         add Poly to $\mathcal{R}$ ;
12         foreach undirected edge $e_f$ corresponding to a directed edge $\overrightarrow{e_m}$
           in Poly do lastVisit($e_f$) = $\overrightarrow{e_m}$, visited($e_f$)++ ;
13     end
14 end
15 return ($\mathcal{R}$, $\mathcal{O}$) ;
```

plete boundary first (borderOfMesh, *line* 4). Every edge there is visited once (*line* 6). Since the first directed edge start is oriented counter-clockwise, all other edges on the exterior boundary are, too. Then the algorithm proceeds further from a mesh's outside to regions in the inside.

The inner while-iteration (*line* 7-13) is understood in the following sense: Until a mesh G_m has been completely discovered, there are still some edges in the mesh which have been visited just once. In other words the number of visits is a direct indicator for the overall progress of the algorithm. Now each of these edges is the starting point for detecting a minimal polygon Poly (*see line* 10).

Edges visited once are, initially, the edges on the exterior border of the mesh. Later, as the algorithm proceeds through the individual regions in the inside of the mesh, the remaining edges e_i with visited(e_i) = 1 indicate that an inner region has not been explored yet. In a second visit the edges are traversed in reverse order, unless they are on the exterior border.

As soon as the exploration of a mesh is finished, the outer while-iteration (*cf. line* 2-14) comes into play: if there is a further mesh in the floor plan graph, none of its edges have yet been visited (due to the lack of direct connection between different meshes). Fig. 31 for instance, features three different meshes.

The algorithm relies on the topological property that an edge divides space into two half-regions. Consequently if an edge has been visited twice, all its neighbouring regions have been taken into account. The algorithm is therefore complete in this regard.[11] Indeed, each edge is visited twice in the end. So this means for the time complexity that it amounts to no more than $\mathcal{O}(2\ E_f)$ (i.e. linear in the number of edges), despite the two nested while-iterations rather indicating quadratic complexity at first glance. In terms of space, the number of visits and the order during the last visit have to be basically stored for every edge.

However, there is an additional time complexity of $\mathcal{O}(N\ deg_{avg}$ $\log(deg_{avg}))$[12] in the pre-processing step of Alg. 1. This is the complexity required for sorting the incident edges of every node in counter-clockwise order (in order to determine the next edge within a polygon).

In student work supervised by the author, a first version of this algorithm has been implemented along with the basic geometrical model.

ENHANCEMENTS. The basic algorithm presented so far is not suitable for any arbitrary kind of floor plan graphs yet. We have encountered some specific problems in connection with the creation of polygons – for some cases occurring in real floor plans there is no corresponding representation as a polygon. Before we present a systematic categorisation of these problematic cases let us first give some concrete examples:

Consider again the floor plan graphs depicted in Fig. 26. In the coarse graph one edge is 'sagging' in the sense of being oriented towards the *interior* the region (i.e. the edge has the same face on both sides). Similar situations can frequently be found in museums. Although not explicit, these kinds of free-standing walls often subdivide a room (cf. Sect. 5.4.1 for more examples of this specific phenomenon).

A further example for problematic 'thin' walls are room dividers (see Fig. 32). They serve as architectural elements in various cultures, for instance in Spain and Japan. Note that they can be moved, which allows for a flexible subdivision of a room.[13] It therefore makes sense to model them separately from the static structure.

If there were such an inwards-oriented edge of type E_o (opening), it would be considered as self-connection of the region. Modelling an inwards-oriented wall in this way leads to tremendous problems for the basic version of Alg. 1: it stalls because there is *no* consecutive directed edge $\overrightarrow{BC_i}$ to continue; the current directed edge $\overrightarrow{AB}$ is a dead end (this case has been previously omitted).

[11] It is however not complete for all floor plan graphs (i.e. it does not accept all kinds of floor plan graphs as input). In the next section counterexamples are shown. How they can be processed is also discussed there.

[12] Here deg_{avg} is the average number of incident edges per node.

[13] They can, e.g., dock into other elements.

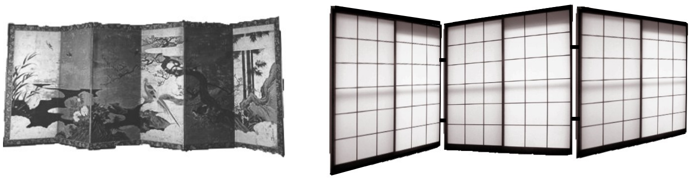


Figure 32: Flexible Room Dividers

As a simple remedy we could extend the algorithm by allowing a sort of backtracking for such situations: the edge $\overline{AB}$ would be visited again in reversed direction. This can be technically implemented by no longer excluding node A from the list of next candidates (i.e. the flipped edge $\overrightarrow{BA}$ is a feasible alternative to choose from[14].).

However, an error still arises when we want to create the resulting polygon: the inwards-oriented edge appears *twice* in the representation as a polygon, while the definition of a polygon (cf. Def. 5.3.3) prohibits any repeating elements.

In order to obtain a proper polygon, these two opposing edges would have to be represented separately. Therefore the points A, B could be shifted slightly into a new edge between A', B'. Next, an edge could be created between B and B' (and between A' and A as well) to avoid repetitions and points with the same coordinates. Every edge having the same region on both sides could in this way be replaced by a small region that represents this boundary.

These two examples give an idea of the problems encountered. In general, neither edges nor single nodes may occur repeatedly in polygon representations. A classification of problematic cases is shown below in Fig. 33. We assume that the basic algorithm has been extended with backtracking:

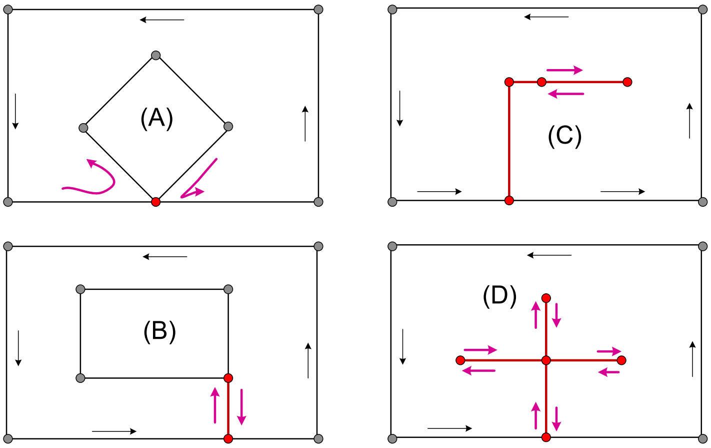


Figure 33: Problematic Cases for Polygonisation

Case (A) shows a single node being traversed twice (marked red). Similarly, the bridge edge described in case (B) is traversed twice

[14] For correctly determining the minimum/maximum angle in the presence of A, choose as an angle of $\overrightarrow{BA}$ 360° and not 0°

(also in red). Note that in contrast to the case of one inwards-oriented edge, these constellations cannot be discovered *immediately* after the first traversal through the node/edge. Case (C) is a sequence of several inwards-oriented edges and case (D) a complex combination of the previous cases.

To avoid this kind of problem it is feasible to choose a fine-scaled modelling which takes into account the thickness a priori (like on the left hand side of Fig. 26). With a coarse-scaled model however, we basically have two options for processing:

- one approach is to relax the conditions for a polygon insofar as to allow loops (repetitions of nodes and edges). This means that the same boundary segment may appear *twice* in one polygon, although in opposite directions. Corners can even appear several times. The resulting list of corners is then of course not a polygon in the *strict* sense of Computational Geometry. Notwithstanding the oddity of the resulting data structure it seems to be useful since the angle between two coinciding boundary lines is well-defined.[15]

  Our first intuition is that this extension does not affect standard algorithms from Computational Geometry such as line intersection. However, a formal proof would be necessary to underpin this assumption (and ensure a safe use of *all* basic algorithms). So one must be careful and check the conditions of every algorithm properly before using it.

- an alternative is to recognise problematic situations in a floor plan graph and transform them into a valid polygon model. Ideally we want to identify these problems on the fly, i.e. while going through the floor plan graph and fix them. For this purpose the previous algorithms have to be extended (as indicated above). It is generally not sufficient to have a look-ahead of just one (peeking the next element), but the problematic situations may arise within a given window, as in cases (A) and (B) of Fig. 33. Therefore, detecting these problematic situations has its own price in terms of efficiency: every time a directed edge is to be added to the polygon, one has to check whether it has been already visited in reverse direction, or at least one of its nodes. This entails not only an augmented memory consumption, but also a worse – quadratic – processing time in the number of edges E_f. This is a typical case of a trade-off: additional expressiveness is bought at the cost of time complexity.

In summary, one can either make sure that only fine models of floor plan graphs are accepted as correct input, or one has to take into account a more sophisticated processing for recognising and handling the problematic cases. The idea of transforming the floor plan graph which has been sketched above on the example of Fig. 26 can be summed up succinctly:

From a practical point of view this means the following:

[15] the angle between these opposing boundary lines is 360°.

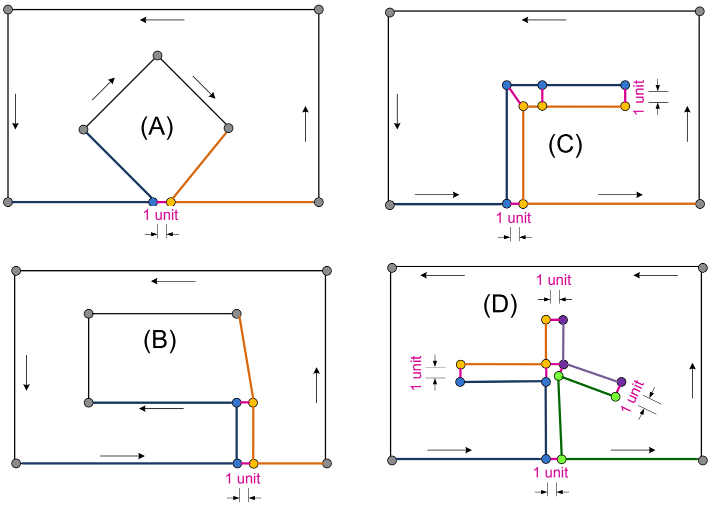


Figure 34: Transforming the Problematic Cases into Valid Polygon Models

- if a node is encountered for the *second time* in the current polygon it has to be split into two nodes so that it does not occur repeatedly in the polygon. To close a possible interior region, the two nodes are connected (see Fig. 34, (A)).

- if an edge $\overline{AB}$ is encountered *twice* – in opposing directions $\overrightarrow{AB}$ and $\overrightarrow{BA}$ – during the creation of one polygon, simply create a new edge $\overline{B'A'}$ for the second traversal. Then, replace the edge $\overline{AB}$ by the polygon $ABB'A'A$ (see Fig. 34, (B)).

  The new edge $\overline{B'A'}$ is shifted by the least possible amount (i.e. basic unit of the spatial reference system) to the right of the first edge $\overrightarrow{AB}$. Then, B is connected to B' and A' to A so that the enclosed wall is modelled as a polygon too. A' is used instead of A for connecting to the next boundary segment AC.

  If AC, again, has been visited once, A' is moved one unit further so that it does not happen to be on $\overline{AC}$ (see Fig. 34, (C)).

A repeated application of these simple concepts is shown in Fig. 34, part (D). Repeated shifting may cause smaller errors with the orientation of the new edges (see (C) and (D)). However, this is almost inevitable when we give walls a certain thickness. We could also shift the original points, but it would be more complicated.

Also, care is required with the creation of new polygons: if, for example, the replaced point A is on the exterior boundary of a mesh, one has to set the counter `visited` for the new connecting edge $A'A$ on the exterior boundary to one.

In summary, we can also deal with non-polygon structures and transform them into valid polygon models. Note that this extension does not come for free – there is a trade-off for the additional expressiveness: in order to determine whether nodes and edges are encountered twice during the construction of a polygon, one has to maintain a list of previously visited nodes and edges during this process (with an additional space complexity of $\mathcal{O}(N_f + E_f)$). For each possible expansion of the polygon, one has to check whether the respective node and edge have already been encountered and proceed in a manner as described above. The task of checking whether the current node and edge are in this list increases time complexity to $\mathcal{O}(N_f + E_f)$ for each step. Thus the overall time complexity amounts to $\mathcal{O}(2\ E_f\ (N_f + E_f))$, losing linear time for polygon construction.

5.3.3 *Basic Mapping to a Flat Graph*

Now that we have a polygon representation for all regions in a floor plan graph, we can shift the focus from modelling a building's structure to modelling wayfinding in a building. First, we can define a mapping to the corresponding dual region graph of a floor plan graph. The dual region graph serves as a basis for elementary navigation tasks such as path finding:

Definition 5.3.4 (**Dual Region Graph**). Given are a planar floor plan graph $G_f = (N_f, E_f := E_b \cup E_o)$, a set $\mathcal{R}$ of polygons for every region enclosed in G_f (such as provided by Alg. 3), and a set $\mathcal{O}$ of polygons for the exterior boundaries of all meshes in G_f. A **dual region graph** $G_\mathcal{R} = (\mathcal{R}, E_\mathcal{R})$ is an undirected *multi*graph whose nodes are polygons $P \in \mathcal{R}$. Edges in $G_\mathcal{R}$ are all openings $e_o = (n_a, n_b) \in E_o$ shared by two different polygons P_i and P_j (i.e. the end nodes n_a and n_b appear in both polygons: $P_i = ..n_a, n_b, ..$ and $P_j = ..n_b, n_a, ..$).

Note that there can be more than one edge between two regions, in case that P_i and P_j share several openings on their common boundary. That is why the dual region graph is defined as a **multigraph** instead of an ordinary graph. The above definition covers the basic topological relation of *adjacency* between two polygons. Adjacency is simply defined as sharing an edge of type E_o (opening). The intention behind a region graph is the following:

- regions represent areas of walkable space (the interior of polygons),
- they are delimited by physical boundaries (the contours of polygons),
- openings (of type E_o) on the boundary allow us to move from one region to another.

Annotation with Context Information

As discussed in Sect. 4.1.3, it is possible to annotate the regions and edges in the dual graph with context-specific information (e.g. for modelling stairs, main entries, ramps, etc.). This context-specific information is stored in an ontology, with the elements in the dual region graph linking to the corresponding instances of the ontology.

Room-to-Room vs. Door-to-Door Navigation

One can primarily use the dual region graph to perform path finding, on a *room-to-room* basis. This kind of path finding is more abstract than, for example, *door-to-door* path finding (e.g. by means of visibility graphs, see Sect. 2.1.3) because the purely topological properties

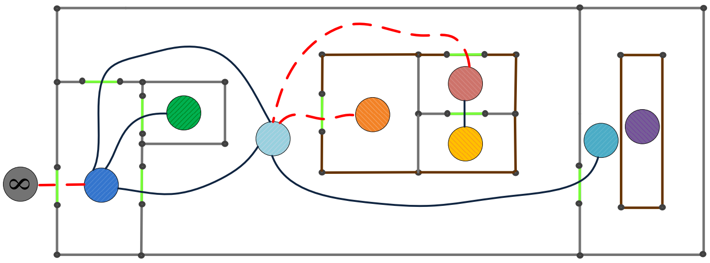


Figure 35: Dual Region Graph for a Floor Plan

of regions (their ordering in a traversal and possible alternatives) are considered *before* looking at the actual geometric distance at all. As a result we do not need to calculate *all* geometric distances, but only those for regions appearing in an abstract path between origin and destination. The representation with dual region graphs is also much more succinct: If we have, say, a region with n openings to other regions, all internal combinations between these openings – a total of $\frac{n(n-1)}{2}$ – would have to be explicitly modelled as edges in a door-to-door graph (see Fig. 36). This representation becomes especially awkward and inefficient for large regions which possess many openings. Therefore we opt for the more compact representation with dual region graphs. Distances between openings are only implicitly modelled (namely as Euclidean distances); they can be extracted from the underlying geometric representation if necessary. Polygons therefore supplement the description of paths by providing geometric details within a region.

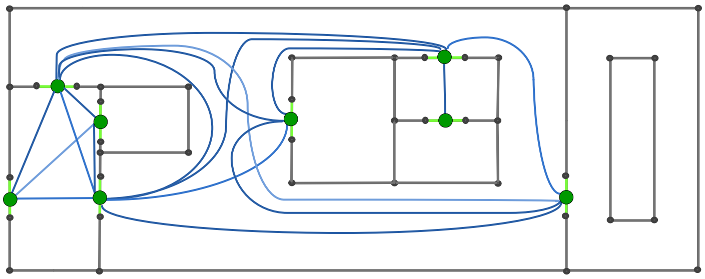


Figure 36: Door-to-Door Graph as Alternative

Nevertheless, two important aspects are still missing in the above definition of a dual region graph (see also the red dashed edges in Fig. 35):

1. the connections along the **inner boundaries** of regions: whenever a polygon (or, more generally, a mesh of polygons) is contained in the interior of another polygon, there are no edges for connecting the containing region to the regions in its interior,

2. the connections on the boundary of the entire floor plan **to the exterior**: these are all openings on the floor plan's outer boundary leading out of the respective floor/building.

In the floor plan shown in Fig. 35, for example, we would expect the

Example

three red dashed edges (leading to the inner rooms, respectively to the exterior face '∞') and the node '∞' to be part of the region graph too. Thus, for the complete construction of a suitable region graph for navigation, one has to take into account the *containment* of regions as well as connections to the exterior. Fortunately, Algorithm 3 provides the necessary means for constructing these additional edges: one can analyse the polygons in $\mathcal{O}$ describing the outer boundaries of all meshes and sort them w.r.t. to containment (the details are explained in the next section).

Reflections

The resulting flat graph, as shown in Fig. 35, would correctly model the topology of a floor plan. However, our intention is to model not only *one* isolated floor plan, but a couple of different floor plans (which make up the whole building) and outdoor spatial networks, too. So let us reconsider the role of these additional elements in a **hierarchical** world model:

Integration with Other Networks: '∞' is Superfluous

The second kind of connections are of particular interest for integration with other maps or spatial networks. All openings which are edges to the infinite outer face represent potential connections to the outside or to other buildings (tunnel, bridge, etc.). Consequently the regions in G_f featuring an opening to the exterior slip into the role of **border nodes** in a hierarchical graph environment (see next section). We can mark them as such. It does not make sense to model the exterior face as an artificial node in a dual region graph given that it is just a placeholder for one or several connecting outdoor spatial networks.

Containment as Hierarchisation

Besides this, the first kind of connections represents a special dependency as well: we know that in order to reach the contained regions, one has to pass through the surrounding region *first* – it is an intermediate goal for such paths. This special relation is not reflected in a flat graph, but we could use a hierarchical representation for this purpose. Modelling containment between different meshes is the first opportunity for hierarchisation *within* a floor plan.

Two Notions of Containment

Note that there are two different notions of containment which have to be distinguished: on the one hand we have containment as a spatial relation between two regions which describe different parts of a building.[16] This kind of containment is captured in the hierarchical model. It is assumed that the structure of the building changes rarely, so this relation is rather **static** (apart from furniture which can be rearranged – these are obstacles at a fine detail level which must be described in a navigation system for blind people). On the other hand containment can also be understood as a relation between the environment and persons or physical objects (such as books in a shelf which are *not* obstacles for wayfinding). Obviously, this is *not* a part-of relationship. The mapping is dynamic, because the location of persons constantly changes while they are moving around.

[16] in the sense that premises are inherently organised into constituent floors, sections, rooms and so forth

The presence and location of persons is therefore not encoded in the graph structure itself, but annotated as attributes of the corresponding nodes (regions) as explained in Sect. 4.1.3. This allows for integrating a positioning system which could update the positional attributes instead of changing the graph structure more easily.

5.3.4 *Hierarchisation*

In this section different forms of hierarchisations are explained for indoor environments. They all make use of the operations defined in the previous chapter. A set of flat region graphs as introduced in the previous sections serves as a basis. We start with containment of regions inside a floor plan. Ways of possible integration with other floor plans, maps of detailed parts, etc. by means of a hierarchical representation are shown. The principles and related problems are illustrated by examples.

5.3.4.1 *Nested Regions in a Floor Plan*

Apart from coinciding boundaries there is another fundamental relation between spatial regions in a floor plan: they can be nested. This aspect is common for environments like parking areas, libraries etc. with many subdivisions inside a region. It calls for a hierarchical representation.

Accordingly, the first step of hierarchisation pertains to nested regions in a floor plan graph. Remember that after applying Algorithm 3 from Sect. 5.3.2.2 one obtains the set $\mathcal{O}$ of mesh boundaries besides the set $\mathcal{R}$ of ordinary region boundaries. Now the challenge is to find out the order in which mesh boundaries in $\mathcal{O}$ are contained in each other. Let us formally introduce a relation for this purpose:

Definition 5.3.5 (Child Relation $\lhd$)**.** Let G_f be a planar floor plan graph with a set of meshes M_{G_f} (their outer boundaries are exactly the polygons in $\mathcal{O}$). A mesh $G_c \in M_{G_f}$ is **child** of another mesh $G_p \in M_{G_f}$, denoted as $G_c \lhd G_p$, if

1. the polygon P_c describing its outer boundary is spatially contained in the polygon P_p of the other mesh's outer boundary (i.e. it lies in the interior of P_p).

2. there is no other mesh $G_i \in M_{G_f}$ which has an outer boundary P_i spatially contained in P_p and, in turn, contains P_c.

Remarks

According to the planarity criterion for floor plan graphs, two different meshes cannot overlap (because the intersection of two boundary segments is always modelled as a node; by definition, the two boundary segments would hence belong to the same mesh). From Definition 5.3.5 and the spatial properties of containment, it follows that the child relation $\lhd$ is both anti-symmetric and irreflexive. The relation defines a partial ordering on spatial regions. Due to the second condition the relation is not transitive. Now this ordering can be used in practice to represent multiple *levels* in a hierarchy, covering the whole spectrum from coarser- to finer-granular spatial regions in a floor plan. The second condition in the definition guarantees that P_p is indeed the *minimal* polygon containing P_c. No other outer boundary P_i of a mesh exists which contains P_c, unless it contains P_a as well. In this sense, the child relation is just a stricter form of containment, namely *direct* containment (representing containment *plus* ordering).

Implementation with Computational Geometry

This is how the implementation, i.e. determining instances of this relation, looks in practice: since all mesh boundaries are uniformly represented as polygons, we can utilise standard techniques from Computational Geometry to test containment for a pair of polygons.

Definition 5.3.6 (Interior, Crossing Number)**. interior**(P) := all points $p = (x, y)$ with an odd **crossing number**. Here the crossing number counts the number of times a ray starting from p (in any arbitrary direction) intersects boundary segments of P.

Ray Crossing

The ray crossing method allows us to determine whether a point p lies in the *interior* of a polygon P. We can use any point on the boundary of a mesh for p. The boundary of a second mesh is the polygon P. Sorting the set of all outer boundaries of meshes appropriate to $\lhd$ takes $\mathcal{O}(|M_{G_f}| \log |M_{G_f}|)$ time on average.

Final Hierarchisation: Subgraph for G_c

Having determined all child relations $G_c \lhd G_p$ this way, one can also find out which of the regions R in the mesh G_p contains the mesh G_c in an analogous manner. In the example of Fig. 35, the region represented as a light blue node contains the mesh G_c with the regions represented as orange, red, and yellow nodes. We can hierarchise all regions in the contained mesh G_c: they form a new subgraph which is spatially contained in the region R. A new node substitutes this subgraph in the dual region graph (cf. the '*partition subgraph*' operation in Sect. 4.2.6). The regions in the mesh G_c which have an opening on the outer boundary of G_c to the exterior region R (we have determined that the exterior is in this context not the global exterior, but locally the region R) become border nodes in this subgraph. For the outermost meshes G_p in a dual floor plan graph (where there exists no $G_o \in M_{G_f}$ with $G_p \lhd G_o$), the regions which have an opening on the outer boundary of G_p are connected to the global exterior (i.e. represent connections to the artificial outer face '∞'). They likewise become border nodes.

5.3.4.2 *Maps at Different Granularities*

Partial Maps for a Floor

Apart from a *given* nesting of regions (which can be derived from a floor plan graph), there are other use cases for hierarchisation to model indoor environments.

Generalisation

Until now it was assumed that a floor plan is available which contains *all* information to be modelled. This is not always the case: instead of one plan for a particular floor, there may be different plans for different parts of a floor. They could be modelled at different levels of detail or granularity too.

Motivating Examples

For example, one coarse map could be used to model a floor in general, whereas another, more detailed map focuses only a specific part (such as the interior of a library in a university building or a special shop/restaurant in a large mall). The first example is depicted in Fig. 37 below. In fact it is quite likely that the local structure of a library/shop is stored separately from the global layout of the floor. For similar reasons this distribution is as stated in Sect. 4.1.1.

Benefits

In the coarse graph the special part is represented just as a node – this node is a black box due to the lack of knowledge on its internal organisation. Hooking in a specific library plan containing the rows of shelves, e.g. for different categories of books, would be convenient. Hierarchical graphs are quite useful in this respect: the two plans can be combined into just one representation while the structural content of both is preserved. This facilitates a flexible construction of a comprehensive model.

Let us illustrate the integration process using the concrete example of the library:

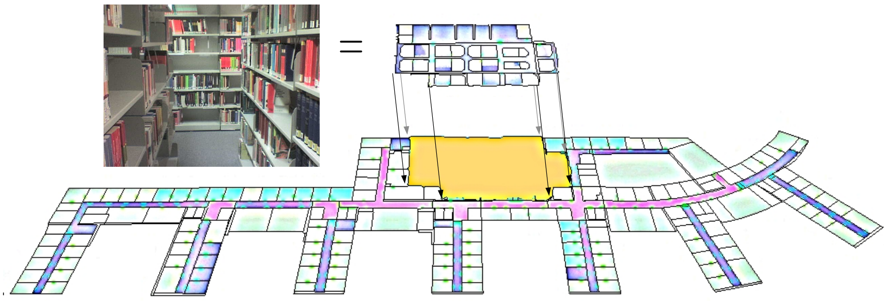


Figure 37: Refining the Interior Structure of a Library

Hooking in a Detailed Map

In the region graph G_l of the library part some nodes have been determined as border nodes (which would connect to the outer face '∞' in case of a flat region graph). These are used in a node refinement operation (see Sect. 4.2.1) which relates the library node n_l in the floor plan graph to the library graph G_l. In order to be compatible, such a border node must exist in G_l for every edge incident to n_l.

5.3.5 *The Third Dimension*

Merging Floor Plans

A larger building normally consists of several floors. So if we want to represent the whole building in one graph, we have to model the transitions between different floors especially. However, it is challenging to model the third dimension. The main problem in practice is simply that we do not have an explicit three-dimensional model of the building (such models are difficult to obtain). Instead the starting point is a collection of two-dimensional floor plans which must be put together. The only three-dimensional objects possibly appearing in these plans (although only represented in two dimensions) are vertical elements such as stairs, ramps, lifts or escalators. How does this extended modelling work in general and in more interesting cases where we encounter, for example, mezzanines or stairs which fork or join at some point (see Fig. 38)? All in all, merging different floor plans is an important subtask in the construction of an indoor model.

Modelling with Hierarchical Graphs

At this point the advantages of a hierarchical graph model come into play: each region graph of a floor plan can be abstracted to a node in a coarser graph (via a node refinement operation). An example of such a coarse graph representing an entire building is presented on the right hand side of Fig. 38. One can then create edges in this coarse graph as connections among different floors. But we are still missing the **border nodes** (cf. the definition in Sect. 4.2.4), i.e. the regions where the new edges in the coarse graph lead into the region graphs of the respective floor plans.

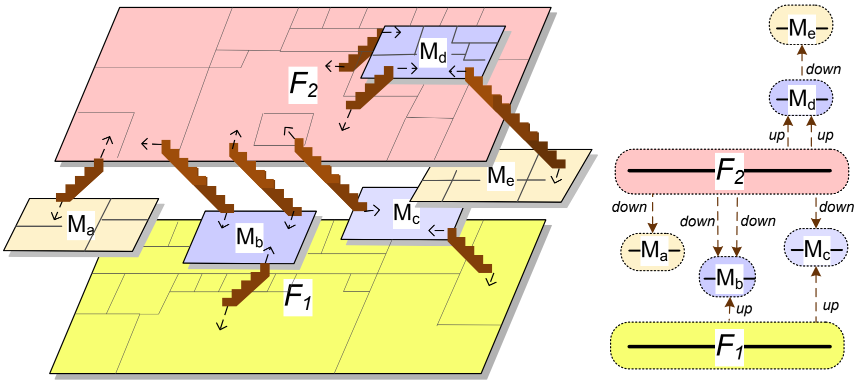


Figure 38: Connections among Floors and Mezzanines in a Hierarchical Graph

Border Nodes

In fact, all special three-dimensional elements within a floor's region graph (these can be *staircases*, *ramps*, *ladders*, *escalators*, *elevators* or even *paternosters* etc.) qualify as potential border nodes to other floors. These vertical elements are modelled as regions in the dual graph of the floor plan. They have to be explicitly annotated with their type in order to be distinguished from the other, non-vertical regions in the graph. Notice that in general, *not all* three-dimensional elements necessarily have to connect to other floors: there might also be exceptions where two regions on the same floor are connected by a vertical element (think of a ramp, for example, in the middle of a corridor or a short flight of a stairs leading to a balcony on the same floor). In this case it is just a matter of ordinary spatial regions with the corresponding type annotated. However, they do not adopt the function of border nodes.

Mezzanines

Apart from floors, one sometimes also encounters mezzanines (see Fig. 38 where $M_a, \ldots, M_e$ represent such). Mezzanines may have an inner structure consisting of several regions. The same algorithms can be applied as for floors to obtain a dual region graph. In general, the region graph of a mezzanine is somewhat smaller than the region graph of an ordinary floor. Whether or not a mezzanine has to be modelled explicitly depends on two factors:

- firstly, if it consists of just one region or more (in the former case we speak of a landing – usually between stairs; such examples are M_b and M_c in Fig. 38). Mezzanines with more than one region are called composite.
- secondly, if the mezzanine has three or more connections to other mezzanines/floors (like M_b and M_d in Fig. 38). Then it is a so-called **decision point** for navigation – the wayfinder can choose among two or more alternatives to continue his/her path.

Referring again to the example of Fig. 38, modelling the mezzanines M_a and M_e is unproblematic. They are both composite, but have only one connection to another floor or mezzanine each. So they are dead-ends. Whether or not to integrate them into the corresponding floor / mezzanine (M_a to F_2 and M_e to M_d) is just a matter of taste.[17] However, the other three mezzanines in the example are more problematic, as we shall see:

Which Mezzanines are Relevant

We cannot model the connection from F_1 to F_2 through M_b as an edge in the coarse graph because *three* different stairs are involved. This would require the use of a **hyperedge** as a ternary relation in the coarse graph. In lieu thereof, we introduce M_b as new node in the coarse graph. It has two parallel edges to F_2 (leading to different border nodes), and one to F_1. As a result, solely the mezzanine M_c can be omitted in the modelling of the environment in Fig. 38 because it is a single, non-composite region with two connections only. The other two mezzanines, M_b and M_d, have to be modelled as nodes on their own (assuming that M_e is also explicitly modelled). They consequently appear in the coarse graph for the whole building. From this simple example one can already recognise the main points which matter for wayfinding:

Hierarchical Path Finding

The two floors F_1 and F_2 are composite spatial regions with a rich internal structure. At the level of hierarchy we are now looking at, it is sufficient to view floors as black boxes: When searching for a path from a region r_1 in the dual region graph F_1 to another region r_2 in F_2, three abstract paths from F_1 to F_2 can be found (two via M_b and one via M_c). These three paths are then refined at each of the floors' region graphs, from the corresponding border nodes to r_1 / r_2. These partial paths can be computed in parallel and then subsequently combined into a final result.

5.3.5.1 *Connecting Different Floor Plans*

Partial Ordering

In order to know which region graphs potentially connect to other

[17] Note that there can also be small flights of stairs within the same floor, e.g. the start of a corridor and similar.

region graphs (by means of border nodes), a *partial ordering* relation among floors is additionally required. This relation *qualitatively* describes the height/level of the floors, or more precisely, the order in which the floors are stacked upon each other. Since it is not necessary to compare two floors in altitude – unless they are neighboured, a partial order is sufficient for this purpose. For a concrete route description, as in the example of Fig. 38, one does not need to know whether M_e is actually above or below $F2$ (this cannot be concluded from the two facts 'M_d above $F2$' and 'M_e below M_d' anyway). With this additional knowledge, we can try to automate the integration of different floor plans. The task is to determine which of the vertical elements of neighbouring floors potentially match. There are basically two different ways of doing this:

INTEGRATION BY SPATIAL COORDINATES. One can overlay two floor plans and create connections for all border nodes which are above/below each other. The spatial coordinates of border nodes can be exploited for this purpose (to determine an overlapping). This requires the two spatial reference systems of the two floor plans being aligned.

Same Reference Systems?

However, they do not always coincide if we just take the dimensions of the scanned floor plans as a reference. There could be rotations, use of different scales or parallel translations of the coordinates in the spatial reference systems of the different floor plans. These issues hamper integration. The problem can be solved by bringing all floor plans into a common spatial reference system. For each pair of floor plans one has to know the correspondence of at least three points to transform them into the common spatial reference system (rotation, translation and scaling are operations which can be done by multiplying all coordinates with a suitable transformation matrix[18]).

On the one hand, the manual effort for this method is limited to bringing all floor plans to the common spatial reference system. On the other hand, this method suffers from one crucial problem: **misaligned stairs** which are not directly above or below each other on the respective floors cannot be automatically captured by a two-dimensional approach: The problem is that even though a pair of stairway ends on different floors could be matching (according to spatial proximity), it is still not guaranteed that they really correspond. There could be the case of crosswise stairs, thus the matching end belongs in fact to some other stairs despite spatial distance. In other words, the two-dimensional matching process is, in general, not deterministic and cannot be automatised without knowledge of the three-dimensional structures of the stairs.

[18] The formulae are of the form $\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = M \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$ where x, y are the coordinates of the local reference system, M is a 3×3 matrix, and x', y' are the new coordinates of the common spatial reference system.

INTEGRATION BY SYMBOLIC IDENTIFIER. The alternative approach is to label all stairs and other regions in different floor plans that have vertical connections with the same symbolic identifier. Although this method requires more manual work, it is far more flexible and can cover complicated cases with misaligned elements and also all sorts of distortions. In practice one can also use a combination of both approaches, e.g. use the first one as default unless there are identifiers which do not correspond. Then, one only needs to specify the same identifiers in both floor plans for the problematic cases.

5.3.5.2 *The Role of Vertical Border Nodes*

Now let us take a closer look at the different types of border nodes. A preliminary classification of different staircase types is depicted in Fig. 39. Note that this classification is fairly general. Most staircases fall into the small set of these categories or are at least a combination thereof. Similar shapes can also be found for escalators, elevators and other three-dimensional elements.

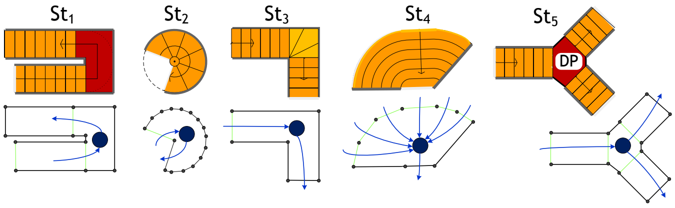


Figure 39: Different Types of Staircases

Classification

At first glance, the geometries of the stairs seem quite different: Whereas the staircase St$_2$ is bounded in the inner part by itself due to its winding structure, the staircase St$_4$ is semi-open. It offers quite a number of entry points at different angles. What all these examples have in common is the fact that they are closed to the lateral (by boundaries such as rails, walls, or simply by a height difference which cannot be overcome).

Implications for Navigation

Notice an interesting aspect for navigation: the staircase types St$_1$ through St$_4$ possess only one entry and one exit point, each. Since these staircases afford movement only *along* their structure, their geometry is totally irrelevant for wayfinding, despite its possible complexity (e.g. zig-zag stairs with several non-branching landings). However, the staircase type St$_5$ shown in Fig. 39 is atypical: it has a landing with three different flights of stairs branching off. Coming from a certain direction, there is always an alternative for movement from the landing. This consequently makes the landing a **decision point** – the wayfinder has a choice on which branch to continue[19]. Hence this landing cannot be combined with the stairs to a linear structure. It has to be modelled as an individual region.

Example

We want to illustrate this concept using a concrete example: Fig. 40

[19] A 'branch' in this sense does not necessarily have to be a vertical connection, e.g. of the kind `stairs`, but it could also be a `door` leading to another region.

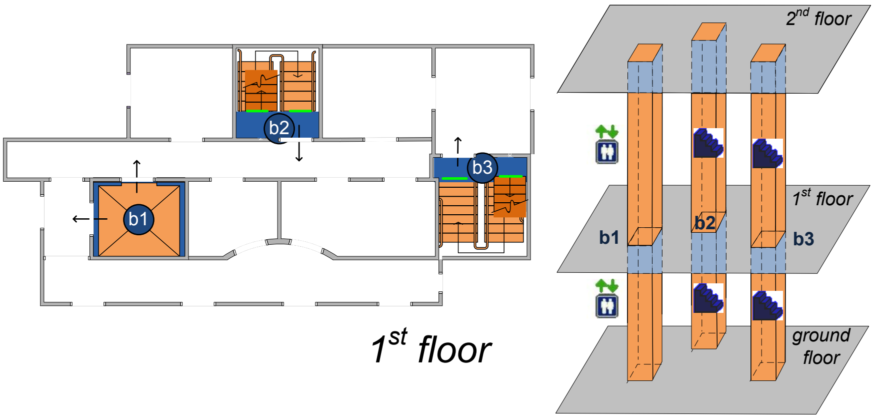


Figure 40: Integrating Floor Plans

shows a cross section through a floor, with b1, b2, and b3 being border nodes. It is striking that the elevator b1 has two connections to other regions on the first floor. Besides this, an elevator can of course move up and down (escalators and paternosters can be modelled in a similar way). This is modelled as a series of linear connections among floors with the respective border nodes being regions similar to b1. Following an abstract path in the coarse graph with these connections can be wrapped up – it corresponds to the single instruction "take the elevator to the n^{th} floor". The other vertical elements on the first floor, namely b2 and b3, are landings of staircases which have upwards and downwards connections (in the coarse graph) and an ordinary connection to another region on the first floor. Unless there are further branchings of stairs between two floors, the stairs can be simplified to linear structures. They thus become edges in the coarse graph.

5.3.5.3 *Connected Components*

The third dimension (and the other forms of hierarchisation as well) implicate specific problems. As aptly pointed out by Duerr [BD05], hierarchisations of certain buildings may not necessarily be *unique*. Fig. 41 demonstrates such a case:

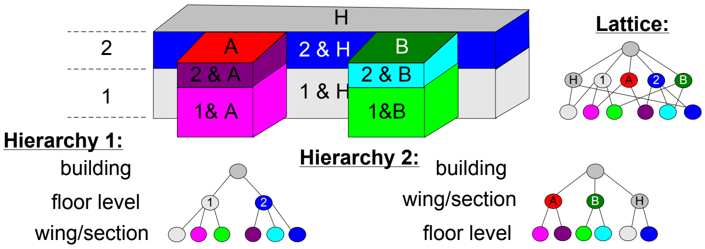


Figure 41: Two Different Hierarchisations with Floors and Wings

Example 1 This particular example shows an attached part of a building, i.e. a wing stretching over several floors. It intersects with more than one floor. The resulting hierarchy is not a tree but a lattice. Consequently two different hierarchisation strategies are possible: either one models the wings *first* and then represents the several floors within the building parts or conversely one decides to model the floors as most abstract levels and *then* distributes wings and sections across floors on the different floors.

Awkward Navigation

Moreover, the hierarchy always depends on the concrete structure of floor plans. This structure however, does not necessarily reflect the *navigational structure*. Some reorganisation may be required to obtain a suitable navigation structure:

Example 2

Consider the example of the building illustrated in Fig. 42. It has three towers on the upper floor levels: in this particular case the simple subdivision into floors alone is not very practical – it has nothing to do with the actual navigation in the building. Imagine for instance, two rooms R_1, R_2 on the same floor, although in different towers. For reaching R_1 from R_2 or vice versa, a detour through other floors is required. It is apparently more sensible to consider the three towers separately because they are disconnected on the upper floor levels. If the region graph of a particular floor turns out to be disconnected, it has to be broken down further into its connected components.

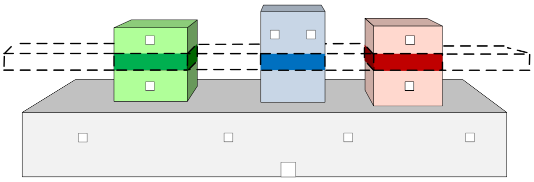


Figure 42: Separate Towers on the same Floor Level

Example 3

This example is by no means unique. In train stations we might encounter a very similar situation: the section where the trains arrive and depart is typically divided into several platforms. Although all platforms are on the same floor level, they are separated from each other by rail tracks and it is forbidden to walk over these. In a number of buildings the solution chosen by the architects is to introduce a special transit area on another floor (e.g. a tunnel or bridge) which connects all of these platforms. An example is depicted in Fig. 43. To get from one platform to another (e.g. when changing trains for transit) one has to take vertical connections such as escalators or a lift and pass through the transit area.

Résumé

Many other similar examples spring to mind, e.g. opera houses with several balconies or building complexes connected by bridges. All these examples suggest that the *internal* connectivity of graphs is indeed a key point for the automated construction and use of a graph hierarchy.

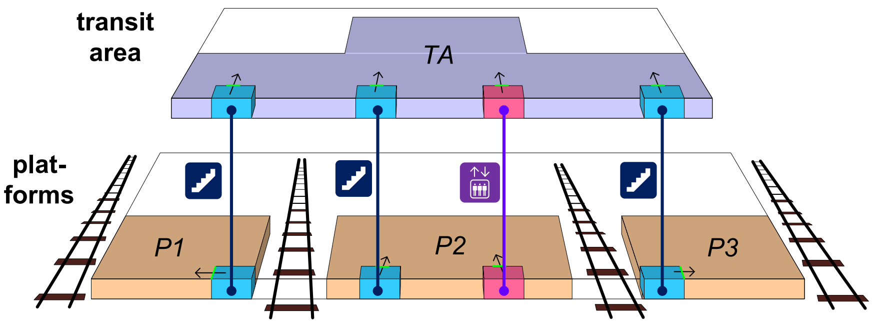


Figure 43: Train Station: Separate Platforms Connected by a Transit Area

5.3.6 *Summary*

The main contributions of this section, in a nutshell are:

- a simple geometric model for planar floor plan environments has been proposed in Sect. 5.3.2.1. It is based on only three primitives, namely **spatial regions**, their boundaries, and special **openings** on their boundary. Modelling boundaries and openings alone is sufficient: regions are implicitly defined; they can be derived from the boundary structure (see Sect. 5.3.2.2).
- Sect. 5.3.3 and 5.3.4.1 explained a method to map the elements of the geometric model to a dual graph structure. In this graph structure regions are represented as nodes and openings as edges between two regions.
- In general, the model needs to be computed only once.[20] This is very convenient, because it can be handled in a pre-processing step. All static aspects are covered there.
- Different ways of hierarchisation have been investigated in Sect. 5.3.4 for dealing with the three-dimensional structure of buildings, integrating plans at different levels of granularity and modelling containment of regions within a floor plan.

[20] except for dynamic aspects which are subject to change, such as rearranged furniture

5.4 FURTHER ENHANCEMENTS OF THE HIERARCHY

With the systematic method presented in the previous section, a hierarchical graph structure can be constructed from a given set of floor plans, step by step. This hierarchical graph serves as an aid for indoor navigation tasks. Notably, in the single graphs of a floor plan, every spatial region is represented by exactly one node, and every opening on a region's boundary by exactly one edge – thus, there is a direct one-to-one correspondence between graph and floor plan.

5.4.1 *Insufficiency of the Basic Model for Real Navigation Problems*

The insufficiency of this basic hierarchical model for real navigation tasks and guidance of pedestrians in buildings is to be demonstrated in the following – some specific, yet relevant cases cannot be handled properly. Practical experience has been gained from an analysis of spatial regions in real buildings. Some spatial regions expose typical features which make wayfinding more difficult. The problematic cases encountered in this analysis are now highlighted and illustrated by examples:

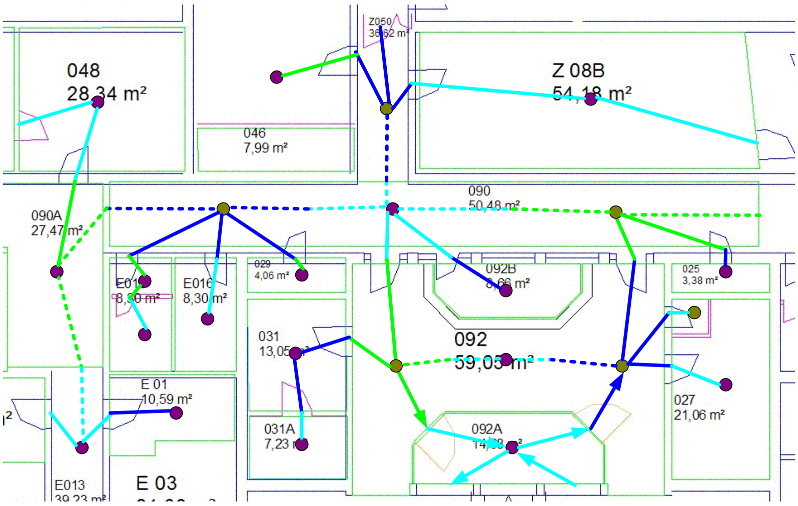


Figure 44: Example of a Manually Overlaid Floor Plan

Difference to an Intuitive Graph

Consider for instance the excerpt of a floor plan shown in Fig. 44. It has been manually overlaid with a graph structure to facilitate navigation. Remarkably, there is *not* always a one-to-one correspondence between a node and a spatial region. The corridor structure for example, has been divided into three nodes because of its length. The same holds for the entrance hall, because of its irregular shape. As it turns out, in this example nodes are used for describing parts or reference positions within a spatial region. As these nodes have been chosen by humans, one may be inclined to say their placement is just arbitrary or random. However, this is not quite true. There are some principles guiding such a decision. For example, all reference nodes represent points which are visible from each other.

Getting behind the Reasons

Obviously, it is quite inconvenient and tedious to overlay a large set of floor plans manually with a graph structure. We therefore want to alleviate this problem and propose a systematic method for obtaining these nodes automatically. In conclusion, the approach presented so far has some inherent limitations. For enhancing the basic model, this means appropriately answering the question: "When should a node for a spatial region be split?". We now want to identify the special cases that cause these problems and systematically analyse them.

Example: Maze

A maze like in Fig. 45 is the prototypical example for an environment which consists of only one spatial region but nevertheless is very complex from the point of view of a wayfinder: one has to make several decisions in the interior of the region to traverse it. These decisions pertain to the question of how to continue a path through the region. Some may lead to dead ends while others are 'right' for the purpose of getting out of the labyrinth. Reference points where decisions are made are highlighted in Fig. 45.

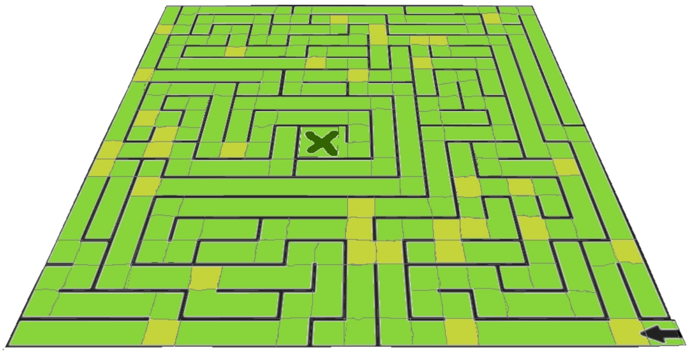


Figure 45: A Maze as an Example for a Complex Spatial Region

Implicit Decision Points

We can see that the special structure of this spatial region has an influence on the choice of paths: the free space where persons can move is, to a large extent, delimited by interior walls. Notice that the boundary is therefore highly irregular and non-convex. One particular problem besides limited freedom of movement is limited advance visibility, e.g. due to walls obstructing sight. Thus a wayfinder has to make uninformed decisions, without knowing whether a dead end might be just around the corner or not. In a way this case is similar to that of museums, although admittedly more extreme. The structure and position of these walls have to be analysed in order to give navigational advice. Because these walls do not end properly, they can be continued in one's imagination until they meet another wall. These extended boundary lines can be thought of as additional openings implied by the structure. Albeit not explicitly modelled, their presence follows inherently from the shape of an interior wall.

Hard vs. Soft Boundaries

This argument is supported by other research in this area: Bittner [Bit01] for example, distinguishes ontologically between two kinds of boundary. The first are impenetrable, hard (*bona-fide*) boundaries like walls. They are explicitly modelled. However, as Bittner argues, there is additionally a second kind of boundaries. These are soft (*fiat*) boundaries. Notwithstanding their invisibility, they exist and are often unconsciously passed by the wayfinder (unless they are *explicitly* referred to in navigational instructions). Anticipating these implicit decision points from the structure of a spatial region is a very desirable property for an automated approach. In following section, we take up the notion of soft boundaries developed by Bittner and incorporate them into a geometric decomposition method for spatial regions. The role of soft boundaries in the graph model is explained there.

Apart from this, there are many further examples of problematic situations:

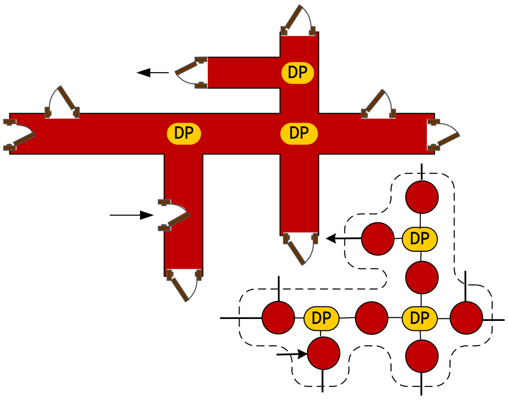


Figure 46: Implicit Intersection of Corridors

- from the corridor system depicted in Fig. 46, only one spatial region is extracted according to our basic approach. In spite of this, a person would not intuitively classify this space as only one region. One can easily see that the inner structure is rather complex: there are several *implicit* intersections in this region. They are found where perpendicular corridors cross each other (marked by 'DP' in Fig. 46).

  From a geometrical perspective, each intersection is described by four non-convex corners. This implicit subdivision also has consequences for a wayfinder: Generally speaking, several instructions may be required for going through this complex spatial region. Wayfinders have to make *decisions* at each intersection (labelled 'DP') because there is a choice how to continue a path.[21] These intersections could hence be suitable candidates for landmarks [Lyn60, Gol99] in a path description.

- the example of Fig. 47 demonstrates that walkable space in the interior of a spatial region can be delimited by inner obstacles as well. These obstacles do not constitute the boundary of the

[21] so even when their path continues straight ahead, this is already a decision.

region itself. They are represented as independent polygons. In particular, the 'spatial region in spatial region' constellation is shown in Fig. 47:

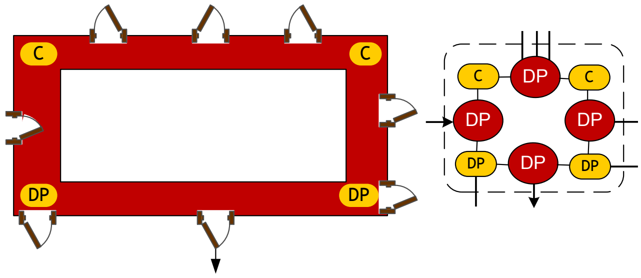


Figure 47: Circular Corridor due to Inner Obstacle

The circular structure results directly from the containment (i.e. child relation) of spatial regions[22] in the concrete example. There are basically two ways for going *around* the obstacle. Hence four corners are implicitly defined. Note that the inner boundary may belong to a composite spatial region, e.g. if several adjoining rooms are enclosed by the circular corridor. They form a mesh. However, Algorithm 3 in Sect. 5.3.2.2 is capable of handling this case as well: the outer `while`-iteration finds all composite spatial regions which are meshes, i.e. connected components, in a planar floor plan graph.

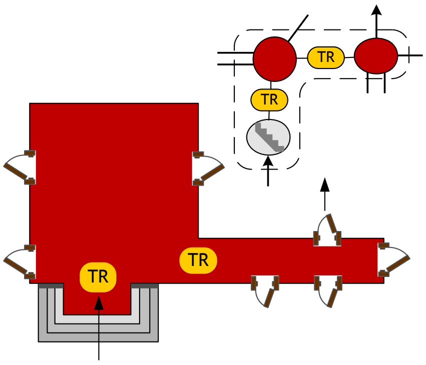


Figure 48: Implicit Boundary between Hall and Corridor

- Fig. 48 sketches a somewhat different constellation compared to the previous example: again, we only have one spatial region. But there is an **implicit boundary** between the large hall on the left hand side and the narrow corridor structure on the right hand side. The corridor seamlessly runs into the hall. Thus the character of the space changes tremendously (namely from open to closed space) without being explicitly modelled. From the point of view of a wayfinder, this threshold would again qualify as a suitable landmark [Lyn60, Gol99] for a path description.

[22] more precisely, their meshes

- The next example is taken directly from a real building. It shows an atrium from several perspectives (see Fig. 49). The difficulty in this case is due mainly to the third dimension. It lies in the automatic distinction between the central part and its lateral wings:

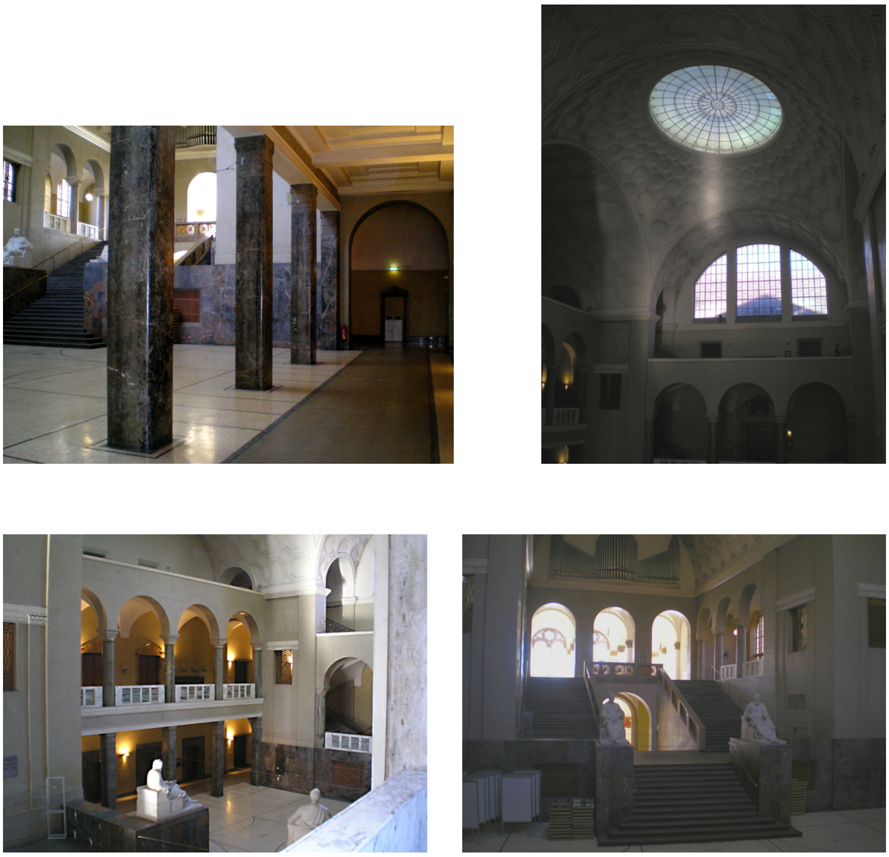


Figure 49: An Atrium from Multiple Perspectives

While the central part rises very high (to the uppermost floor), the **ceiling height** in the lateral parts is considerably lower. There are actually several archways stacked upon each other. On the ground floor however, the whole space is modelled as one spatial region. People may not perceive it as one region however, because of the drastic height difference. This aspect should be reflected in the graph model for better guidance.

Yet again we have *soft boundaries* between the different parts. They are not necessarily modelled but can be implicit. In this case, arches and pillars delimit the separation of space. The ceiling height is an important factor to consider in other buildings as well. Especially in churches it implicitly separates parts of a large spatial region. Similar situations can be found in large opera halls with balconies and other environments with comparably large halls.

- Finally, another problem with regard to visibility can be a segmentation and distortion of spatial regions like corridors. The corridor structure in Fig. 50 highlights both aspects. Such a segmentation may be performed for special reasons. Recognising a 'main corridor' structure, as required for an instruction like "follow the *main corridor* until.." is not possible ad hoc. Several spatial regions constitute the main corridor structure. In this particular case they are separated by emergency fire doors.[23] Nevertheless, it is perceived by people as just one long corridor extending over several spatial regions.

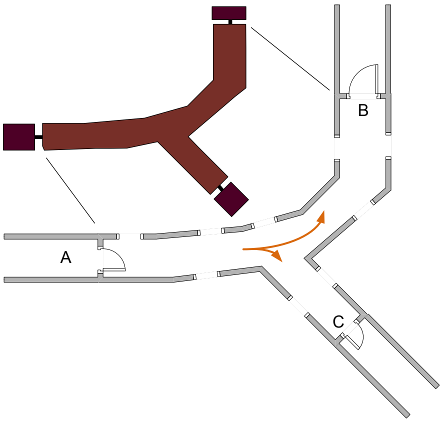


Figure 50: Problem: Segmented Corridor

In contrast to the previous examples, the spatial regions representing the segments are too small to capture the notion of a main corridor. A grouping/clustering of adjoining spatial regions would make sense in this case. Thus, the main corridor could be modelled as a composite spatial region, i.e. subgraph, consisting of several connected segments. Determining the 'rank' of the corridor, i.e. whether it qualifies as a main- or side corridor, is another problem. Global knowledge of the entire floor or even building is required for this purpose, including all entries and exits. This rank may be determined by using a global sort of navigational hierarchy for a building.

In the example of Fig. 50, there is in addition an implicit intersection in one region and, even more problematic, a distortion in the corridor shape. The distortion may lead to wrong orientations in path descriptions, because the spatial relations 'left' and 'right' are relative and change when the corridor is bent to the left. An instruction may state that the door is on the left hand side, whereas in reality it is on the right hand side. One has to take into account that the orientation changes gradually while walking along the curved corridor. Since curved shapes are approximated by polygons and polylines, we generally have a concave chain (see Def. 5.4.1) on one side.

Moreover, there can be ambiguous situations for descriptions: for instance, when advised to "follow the (main) corridor till the end" it may be unclear whether a corridor continues

– around an orthogonal corner or ends at the door in front (see Fig.47).

[23] Building such doors may be a requirement for long corridors to prevent the spread of a fire.

- to the left or to the right hand side at a Y-shaped fork (the intersection in Fig. 50 is problematic in this respect).

Each of these cases describes an 'end-of-corridor problem' where decision points are involved. Note that the continuation of a linear feature along the line of sight may depend on the direction one comes from. In the example of Fig 50, the path from B to A is clearly unproblematic (one may easily oversee C, which is oriented to the back). However, this is not the case for the opposite direction from A to B.

All in all there is an interesting link between human instructions and the grouping of conceptual spaces. Investigations such as performed by Look et al. [LKLS05] shed light into the nature of route descriptions and explain the underlying spatial concepts which can be extracted from a large set of descriptions. In Chap. 7, research on this topic is laid out in more detail.

Concluding, there are some problematic cases for navigation which are not yet covered by the hierarchical model defined so far. One of the key issues is that automatically extracted spatial regions do not always match with a human notion of a room or corridor, due to their complex structure. Two different reasons are primarily responsible for this complexity:

1. a spatial region has some **non-convex** corner (see Sect. 5.4.2) or introversive, i.e. free-standing wall.
2. another child region is **nested** in it.

Soft vs. Hard Boundaries

The special geometry of these regions entails some *implicit* openings. Unlike doors, these openings are not always explicitly modelled. These new kind of openings are called **soft boundaries** [Bit01]. They often fulfill the function of a landmark in a path description.[24]

Extension to Hierarchical Graph Construction

The challenge now is to devise methods which cater for these complex cases as well. An automated method inspired by a human's understanding of space is particularly difficult to find. For this purpose, the idea is to bring hierarchical graphs into play again: on the one hand, complex spatial regions should be decomposed into meaningful parts around inner landmarks. This would yield a more detailed graph for navigating a region's inner structure, but only if necessary. On the other hand, several spatial regions should be grouped together if they form a higher-level concept like a main corridor, section or wing. This can be of use for giving concise path descriptions (referring to these higher-level elements). By describing these structures as composite subgraphs of spatial regions, we obtain an additional hierarchy level between floors and basic spatial regions. So it makes sense to take

[24] In contrast to outdoor environments, indoor areas often lack persistent landmarks which are widely visible (such as television towers). Furniture, often subject to change, is not appropriate for reference either. Instead, salient features in the architecture qualify as natural landmarks. Note however, that inner landmarks have a lower range in general. But on the other hand they are very specific for a local decision.

the dual graph model from a floor plan as a basis and adapt it into a suitable hierarchical model by adding additional hierarchy levels for

- the interior of problematic regions (w.r.t. descriptions) with a complex structure,
- subgraphs which represent coherent, meaningful parts of a floor plan (such as sections or wings).

5.4.2 *Geometric Decomposition of Complex Regions*

Motivation

Until now, the construction of the hierarchical model has reflected a building's topology: this included aspects like the division into different maps and floor plans as well as containment of different regions. Nevertheless – as demonstrated by the previous examples – an important aspect is still missing in the hierarchisation: few assumptions have been made on how people move through a particular region, i.e. paths *inside* a region. For practical navigation tasks, the finest resolution of the hierarchy (that of leaf regions) can thus still be too coarse. Sometimes several instructions are required to describe paths through complex regions.[25] Guidance through these regions by reference positions and additional landmarks makes wayfinding considerably easier for people. The bottom line is that the hierarchical representation needs to be extended, by an additional level comprising more detailed graphs for the inner structure of complex leaf regions.

Cell Decomposition

Visibility graphs and Generalised Voronoi graphs (cf. Sect. 2.1.3) are valid approaches for dealing with this problem. The shortcomings of both approaches – in particular with regard to route descriptions – have been pointed out in Sect. 2.1.3. Instead a different solution is proposed based on the regions' geometry. It falls into the category of **exact cell decomposition** methods [Lat90]. An earlier version of the method presented has been proposed by the author in a research paper [SLO07] but has been revised since then [OS08]. This decomposition method fits nicely into the hierarchical graph model: First and foremost all regions are represented as polygons. Note that the polygons representing problematic regions expose certain characteristics: they are either non-convex or contain another polygon in their interior or both.

Definition 5.4.1 ((Non-)Convex Polygon, Concave Corner, -Chain). A corner C_x of a polygon P enclosing an **internal angle** $\alpha(C_x) := \measuredangle(C_{x-1}C_xC_{x+1})$ greater than $180°(\pi)$ between the two boundary segments $\overline{C_{x-1}C_x}$, $\overline{C_xC_{x+1}}$ is called **concave/non-convex**.[26] A **non-convex polygon** has at least *one* concave corner, otherwise the polygon is **convex**. **Concave chains** are defined as maximum-length sequences $C_xC_{x+1}..C_{x+k}$ of k consecutive concave corners.

What Non-Convexity Means for Navigation

Whereas convex polygons are generally not so critical for route de-

[25] According to the qualitative classification of spaces by Montello [Mon93], these regions are medium spaces: people can only experience them by locomotion and integration of multiple vantage points.

[26] In general, angles α with $180° < \alpha < 360°$ are called *reflex angles*.

scriptions (cf. Sect. 5.5.2), one has to treat **non-convex polygons** separately. The same applies to polygons with inner boundaries, i.e. nested polygons in their interior. This is due to the fact that humans cannot see (in advance) what lies beyond a corner. They must go past/around the corner and reorient themselves to get a picture of the remaining part of the region [Mon93]. E.g., there could be other doors or openings around the corner. The respective corner is then relevant – it can serve as a reference point in a path description, indicating a required turn action.

Off-the-Shelf Algorithms for Decomposition?

Fortunately, Computational Geometry already provides a number of algorithms for decomposing non-convex polygons into a set of convex polygons. Most of them are based on triangulation. After applying one of these algorithms one obtains a set of convex polygons which decompose a given non-convex polygon. However, it is questionable whether the decomposed regions are indeed *meaningful* units, feasible for path descriptions. Triangles and other decompositions can be rather arbitrary in shape; they often do not reflect an *intuitive* decomposition. It is admittedly rather difficult to define the exact meaning of 'intuitive' for a human wayfinder (it can depend on many factors such as gender, background knowledge, spatial orientation skills, etc.). A pragmatic viewpoint is therefore adopted in this thesis, stating that a decomposition is intuitive if it can be used as a reference for path descriptions (see Sect. 5.5.2 for details).

Principles of Gestalt Theory

Some basic principles or guidelines can nonetheless be found in other research areas like human spatial cognition and cognitive psychology: One can assess the 'intuitiveness' of a certain decomposition by adhering to principles of **Gestalt theory** [Wer23, Kof35], called 'laws of perceptual organisation'. This theory considers the peculiarities of human perception of spatial objects and shapes. It is largely based on empirical studies and observations, investigating different phenomena (see Fig. 51). Among these principles, two are particularly interesting in connection with intuitive decompositions:

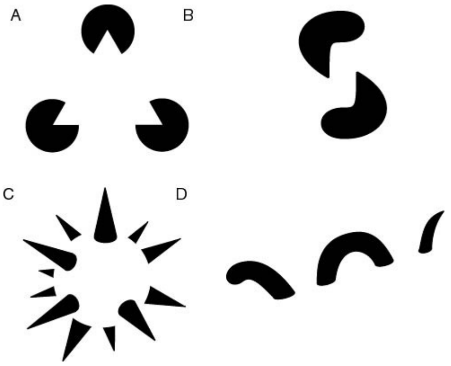


Figure 51: Human Perception goes beyond Explicit Spatial Information (from Wikipedia)

- as the examples in Fig. 51 suggest, the human mind often fills in *missing* information – by extending contour lines – in order to close uncompleted shapes (*closure*) and integrate them into a larger composite shape,

- objects which are close together are perceived as one group (*proximity*). In contrast, objects which are farther away are not perceived as a coherent unit.

Sketch of Practical Application

These principles can be exploited in practice, for a simple decomposition algorithm: first, non-convex corners C_x of a polygon P can be classified according to the degree of their non-convexity (i.e., ranges of the angle $\alpha(C_x)$). Consider the examples in Fig. 52 for this purpose: in the examples 1 and 2 we have what would be intuitively understood as 'corners'. The angle $\alpha(C_x)$ is around 270° in these cases. According to the first principle one can extend the boundary lines $\overline{C_{x-1}C_x}$ and $\overline{C_xC_{x+1}}$ in the direction of C_x (indicated as black, dashed arrows in Fig. 52), until another boundary line is hit.

Concave Chain

Example 3 shows a somewhat different situation: the angles are much smaller, ranging between 180° and 225°. They appear in a concave chain. Extending these lines would not be intuitive because the more the angle converges towards 180°, the less C_x becomes apparent; it gradually loses the quality of a structural landmark. Just around 180°, C_x is in fact an arbitrary point on one continuous or slightly curved wall which goes through C_x. Instead, the line **perpendicular** to $\overline{C_{x-1}C_{x+1}}$ through C_x indicates a threshold or a soft curve at this point. These lines are needed for practical purposes, to describe gradual changes in the intrinsic orientation of the region (an example is given in Fig. 50 in Sect. 5.4.1). Concave chains of such reflex angles just above 180° often describe polygonal approximations of curved shapes.

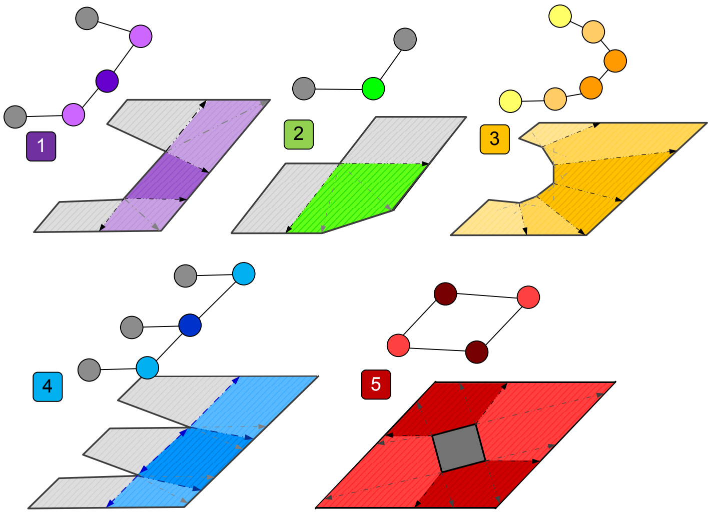


Figure 52: Different Classes of Non-Convex Corners

Perpendicular Extension

The other extreme case is shown in Fig. 52 as number 4, where the angle $\alpha(C_x)$ lies in the range between 315° and 359°.[27] Approaching

[27] The case of 360° is excluded because we are dealing with proper polygons only (see Sect. 5.3.2.2).

359°, the boundary lines fall together more and more. Thus, the region enclosed between their extension and the other boundaries of the polygon (between the grey arrows) has a smaller surface and eventually would become a line (for this and other degenerate cases, cf. Fig. 54). It makes sense to consider the lines **perpendicular** to $\overline{C_{x-1}C_x}$ respectively $\overline{C_xC_{x+1}}$ as well.

Convex Corners of Obstacles

Finally, example 5 shows a polygon O as an obstacle within another polygon P. In this case, the *convex* corners of the inner polygon O (obstacle) play a role. Should they be extended (gray arrows), or rather the perpendicular be dropped (black arrow) to the corresponding boundary lines of P? Since the perpendicular is shorter and the surrounding polygon is the larger one, the second principle suggests that the perpendicular is more intuitive.

Interference of Different Extension Lines

In general, extension lines of different concave corners can potentially interfere with each other. We want to avoid these situations because the intersection of two such implicit lines is not obvious, thus cannot be used as a landmark in route descriptions. Besides this, the inner structure of a region is then fragmented into a lot of small pieces which are rather difficult to describe (see Fig. 53). A simple strategy is to first extend the lines of all non-convex corners ignoring intersections with other extension lines and then resolve these intersections in the following way: for every pair of intersecting lines, remove the longer one (depicted as red lines in Fig. 53). Note that the shorter extension line is more plausible according to the second principle. If too many extension lines have been removed this way, not all decomposed regions are convex. Then the remaining non-convex corners (or convex corners of obstacles) are connected to opposite corners, avoiding intersections with other extension lines.

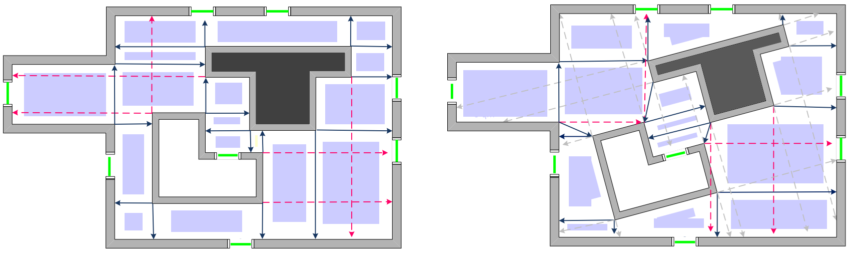


Figure 53: Exemplary Decomposition of a Complex Region into Convex Parts

Interpretation of Decomposed Regions

Essentially, the same algorithm as in Sect. 5.3.2.2 can be applied to a single region (instead of a whole floor plan) once a decomposition has been determined. The corresponding dual region graphs are shown in Fig. 52 along with the geometry. They can be determined in a similar fashion as described in Sect. 5.3.3. As a result decomposed regions are represented as nodes and extension lines between two such regions as edges. Apart from openings on boundaries, the extension lines are interpreted as a second kind of trespassable elements.

The decomposed regions, nodes in the dual graph of the region's interior, serve as reference points inside the region. Junctions between corridors and other examples were shown in the previous section.

Visibility and Dual Graph

In terms of visibility, there is an interesting relation with the dual graph. Two points p_i and p_j inside polygons P_i, respectively P_j resulting from the decomposition are mutually visible only if their direct connection $\overline{P_i P_j}$ intersects *all* extension lines which are edges on the shortest path from P_i to P_j in the dual graph. Otherwise, some extension line has to be crossed in order to see p_j from p_i (or vice versa). While this property holds for every decomposition which is convex, an intuitive decomposition has the additional advantage that reference points and orientations – important for route descriptions – can be fairly easily derived from the decomposed structure. Such a method is outlined in Sect. 5.5.2.

Reference Scale

However, the scale and relative size of some decomposed regions also has an influence on their interpretation and relevance for navigation. Fig. 54 outlines some degenerate cases of decomposed regions. It makes sense to use the size of the human body as a reference scale for the length of extension lines: if an extension line is smaller than this reference scale, it is not passable. However, some smaller objects could be placed in it. Depending on the scale, a small niche in a wall (depicted as red area on the top left of Fig. 54) could either be enterable or not. Obstacles which are located too close to the surrounding walls (bottom right of Fig. 54) likewise block the way through a region.

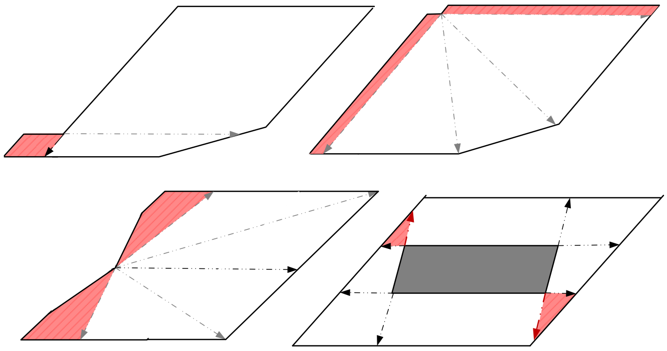


Figure 54: Extreme Cases of Decomposed Regions

In general, if a floor plan is very rich in detail (with numerous small niches, ornaments, and/or obstacles), the reference scale can be utilised to drop all details which are not relevant for human wayfinding tasks. This way the geometry can already be simplified in a pre-processing step.

5.4.3 *Extracting Meaningful Subgraphs from a Dual Region Graph*

5.4.3.1 *Motivation: Practical Use*

Remember that one of the primary incentives for hierarchisation is to allow for **hierarchical planning** (cf. Sect. 5.1). If the structure of a building is reflected in a hierarchy, this can be exploited for giving route directions in a similar way as people do. To clarify the idea, consider following verbal directions given by people:

- *"In order to get to room* D 1.14*, you first have to reach the* D `section`*. It is on the* 1st `floor`*. You can either use the stairs on your right, or if you prefer taking the elevator go straight ahead towards the lobby."*

Looking at the prototypical example from above, several observations can be made:

1. The hierarchical organisation of spaces inside a building is often reflected in the naming scheme (`building` → 1st `floor` → D `section` → `room` D 1.14). Furthermore, it is employed by people for giving directions.
2. There is also a correlated hierarchical order among *decisions*, since wayfinding can be basically seen as spatial problem solving [Pas84]. As a consequence the wayfinding task can be broken down into the subtasks of
   - reaching the first floor,
   - reaching the D-section within the first floor,
   - and finally reaching room D 1.14 from the entrance of the D-section.

   The above line of thought seems to be quite intuitive for people, as a common sense strategy for decomposing spatial problems. Particularly expert wayfinders seem to favour this sort of wayfinding strategy [KTS03] due to its simplicity. The origins of hierarchical planning date back to Sacerdoti [Sac73] (see also the discussion of hierarchical path finding methods in Sect. 3.3).

Text-based Approach

For the automated processing of floor plans, there are a couple of things to keep in mind: Meaningful subgraphs of a dual region graph can often be specified using annotations or labels of graph elements. For example, all rooms starting with a prefix 'D' pertain to a section or wing identified by this label. One can specify patterns as regular expressions on the labels to extract such subgraphs.

Relevance of Nodes

However, not all structures which are useful for route descriptions can be characterised this way: the notion of a 'main corridor' for instance, is usually not explicitly represented. It would be useful to know which regions are more relevant for path finding and which ones are deemed marginal. In other words, the main navigational structure and -axes should be extracted automatically from a given dual region graph of a floor plan. The challenge is to devise a method that automatically identifies the main- and subsidiary navigational axes of an indoor environment and represents them in hierarchical terms: regions with decreasing prominence should also appear in decreasing levels of abstraction.

STRUCTURAL PROMINENCE VS. LANDMARK. Many factors can be found that influence the prominence of an element in the eyes of a wayfinder and, as such, qualify it as a **landmark**: for example, differentiation, singularity, and visual salience determine the suitability as landmark. A peculiar or interesting feature which stands out from the rest of the environment may qualify as a landmark. Variations in illumination might also affect perception. Note that these are mostly *local* properties. Without being explicitly stated in the model, it is virtually impossible to incorporate all these aspects. Therefore the notion of prominence is understood here from a purely topological perspective, complimentary to the classical notion of a landmark. Providing a comprehensive account on the complex topic of landmarks is beyond the scope of this thesis. The interested reader is referred to existing work on the acquisition and selection of landmarks [LTK02, Eli03, EB04, CT05], as well as on the use of landmarks in route directions [RW02, RK04, KW05, HRK06].

Main Navigational Axes

Even if no labels for sections and building parts are provided by the original floor plan data or annotations (which sometimes occurs), one can use the structural knowledge of navigational axes for instructions and also restrict the search space of path finding algorithms. One possibility is to resort to techniques from network analysis, but these yield individual values for certain elements of the network and additionally require a resource-intensive preprocessing. Formation of subgraphs is not evident from this representation. Therefore, a novel technique for forming subgraphs is proposed here. It fits nicely into a hierarchical graph representation. The strengths and limitations of this approach, along with suggestions for further enhancements are discussed in the following passage.

5.4.3.2 *Exploiting the Characteristic Structure of Floor Plans*

Dual region graphs of floor plans are not just arbitrarily shaped graphs – they show a characteristic pattern. The first step is to analyse this pattern of connections between regions, in order to retrace the fundamental wayfinding decisions implied by the structure. The following excerpt of a floor plan (shown in Fig. 55) illustrates a handful of structural phenomena which are characteristical for indoor environments. Most building structures are usually arranged in a similar manner. Chapter 6 gives an account of the results when the method is applied to several floor plans of real buildings.
When we look more closely at the plan, several interesting characteristics for wayfinding are revealed:

- the topological structure of the building in Fig. 55 is rather **sparse**, notwithstanding the two large halls with a remarkable

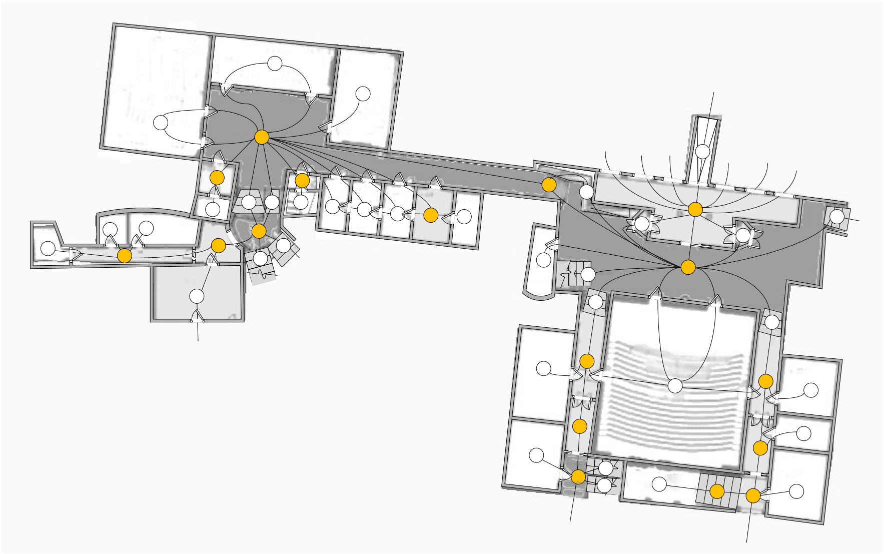


Figure 55: Typical Structure of a Floor Plan (Excerpt)

number of 12, respectively 14 connections – the *average* **degree** of a node (i.e. the average number of neighbours) is relatively small,[28]

- there are rarely **alternative** choices for traversal. Even so, choice appears mostly local (some parallel stairs or exits) since most alternative paths eventually rejoin at a common region (such as the two main halls in the floor plan of Fig. 55),
- whereas the **main navigation structure** should be *public*, some rooms may have only *private* access (see Sect. 4.1.3). Let us assume for instance, that the large room in the middle serves as a lecture hall. Then paths should go *around* rather than *through* the lecture hall. Another indication for private areas can be their secluded position within the building (e.g., the room behind the antechamber). If knowledge about the function of a particular room is not available this may result in an unfeasible path. On the other hand, when this knowledge is available the respective alternative path can be cut off the branches of possible paths by a path finding algorithm.

These characteristics lead to the main hypotheses for the hierarchisation of a dual region graph (forming subgraphs):

NO BACKTRACKING. Floor plans usually contain a substantial number of dead-ends. All these regions have only one adjoining region. For example, the rooms to the left and right of the two corridors in Fig. 55 fall into this category: they are rather secluded because they are only reachable from the respective corridor. Regions which are dead-ends are 'deepest' in the hierarchy in the sense of being merely destinations for wayfinders.

[28] the average degree of a node is below 3 in the floor plans that have been investigated in this thesis. For road networks, it typically ranges between 2 and 5.

If we want to reach a particular destination, there is no need to explore any other dead-ends at all. However, this would be done by a naive graph algorithm.[29] Knowing dead-ends in advance prevents unnecessary exploration, i.e. backtracking for a path finding algorithm.[30]

CENTRAL SKELETON. The previous example suggested that the topological structure of buildings is generally rather sparse. We can conclude that the typical pattern of movement is not like moving from room to room *arbitrarily*, but more often going along main corridors or large halls (figuratively: the skeleton [KTS03] or 'arteries' of a building) *before* entering certain rooms. Obviously, some regions possess quite a number of connections to other regions (e.g. the two large halls and corridors illustrated in Fig. 55). They are also more central in the topological structure insofar as they play a major role in overcoming greater distances and are (on average) more frequently used as intermediate goals for paths. They are represented as colored nodes in Fig. 55. Different wings and sections of a building might be demarcated by these main elements.

Backed up by the two guiding principles, an algorithm is presented which attempts to detect main corridors and halls automatically, distinguishing them from ordinary rooms. It also creates subgraphs of a dual region graph around relevant elements. A first version of the algorithm has been presented by the author at the ACM GIS conference [SSO08]. In the following passage, an improved version of this algorithm is outlined.

5.4.3.3 *Proposed Algorithm*

The principal idea is to partition a dual region graph around 'colored nodes': they express a certain degree of relevance in the graph. How can these nodes be determined? A different measure of relevance is proposed than a measure merely based on a node's degree. It is somewhat similar to **betweenness centrality** (see Chap. 7), but the crucial difference is that one does not have to compute the paths between all pairs of nodes to obtain the measure. A node of a dual region graph in a hierarchy can be prominent for three different reasons:

- it has a connection to the exterior,
- it represents a vertical connection to another floor,
- it is an **articulation point** in the dual region graph (see Fig. 56 and Def. 5.4.2 in the subsequent).

Analysing Criteria

The first two points pertain to connection points for accessing other

[29] A non-informed wayfinding strategy might unfortunately choose one of these rooms and then have to backtrack. Even with using a heuristic such as A*, this can happen if the estimate of the Euclidean shortest distance suggests taking the dead-end.

[30] The structure of a building is rather static, so it is feasible to pre-compute all dead-ends.

parts of a building or leaving the building. Note that they are not *local* properties of a dual region graph. They have already been treated in the hierarchisation as **border nodes** (cf. Sect. 5.3.4). Therefore, the first two points are not considered here again. We concentrate on the last point: **Articulation points** are a concept coming from graph theory; let us briefly recall their definition:

Definition 5.4.2 (Articulation Point). A node $n_{AP} \in N$ of a connected graph $G = (N, E)$ is an **articulation point** iff the removal of n_{AP} from G (including all emanating edges) splits G into two or more connected components.

Previous Occurrences of Articulation Points

We have already encountered articulation points earlier in Sect. 5.3.4.1 without being aware of it: nested regions in a dual graph are only reachable through their surrounding region. Thus the surrounding region is an articulation point in the dual graph. Furthermore, articulation points can occasionally be found in the more abstract graphs between different floors too: Example 2 in Sect. 5.3.5.3 illustrated a building with three towers. The floor level right below the towers is an articulation point and so are all levels of the towers except for the uppermost.

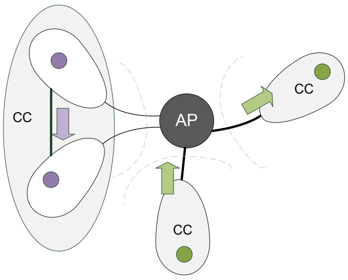


Figure 56: Wayfinding with Articulation Points

Illustration by Example

Fig. 56 shows an articulation point (AP) that would split G into three connected subgraphs (CC). Intuitively, articulation points impose a tree-like decomposition on a graph.[31] This property can be exploited for path finding: if for example we want to go from the lower subgraph 'CC' to the subgraph 'CC' on the right hand side (indicated by the two green arrows on the right hand side of Fig. 56), the node AP *must* be passed – it is an intermediate goal. Additionally, the entire left subgraph can be excluded for this path query. In a second path query (left, lilac arrow in Fig. 56), we want to go from the upper subgraph within the left subgraph to the lower one. Normally the shortest path remains within the left subgraph (CC) unless going through AP would be a shortcut. In any case, the other subgraphs (CC's) need not be explored for this query at all – they can be pruned from search space.

Heuristic

Generally speaking, one can exclude all sub-trees from search which are not between origin and destination. They are, on a higher level, dead-ends for this particular path query. Using this kind of hierarchy

[31] They can also be seen as bottlenecks of pedestrian flows, since different wayfinders with similar destinations all have to pass through them.

makes graph search more intelligent: one takes advantage of knowledge on the spatial organisation (**informed search**) to narrow down search. Thus a decomposition around articulation points is a valuable **heuristic** for guiding path search.

Finding Articulation Points

How can articulation points be found in practice? The basic algorithm is quite simple. Let us sketch the main idea: choose any node $n_{root} \in N$ as the root of a search tree. n_{root} is an articulation point if it has two or more children, i.e. separate sub-trees with no crossing edge between them. Any other node $n_{tree} \neq n_{root}$ in the search tree is an articulation point if it has a child and some sub-tree with no back edge to an ancestor of n_{tree}. The proof of these properties together with a complete linear time algorithm can be found in Tarjan [Tar72] and Gabow [Gab00].

Connected Articulation Points

Especially in sparse graphs, like the dual region graphs of floor plans, many articulation points can be found. Most corridors and hallways fall into this category, since they often have 'attached' rooms which are only reachable through them. Note also that there is rarely a *single* articulation point, but more often an entire chain or graph of *connected* articulation points (like a segmented corridor or corridor system). In case there are many articulation points, their relations need to be examined too. This is useful to for example distinguish between a main corridor and its peripheral wings if all are articulation points. Therefore we distinguish between ordinary articulation points and articulation points which have more than two articulation points as neighbours. The latter are called **decision points** in this context (among articulation points). The other articulation points can be grouped into **chains** (all have two or less neighbours of their kind).

An example with a hierarchisation based on these concepts is presented in Fig. 57, on the previous excerpt of a floor plan. The corresponding algorithm for hierarchisation can be explained as follows (see Alg. 4):

Hierarchisation around Chains and Decision Points

The blue subgraphs in Fig. 57 are formed from chains of articulation points (line 4), merged with connected components (CC, white subgraphs) which are only adjacent to articulations in this chain (line 6). These CC's can be considered as dead-ends generalised to subgraphs. Likewise, such CC's are merged into subgraphs of decision points (line 5). The resulting subgraphs are shown in Fig. 57 as lilac subgraphs. If the decision point on the left hand side were only an articulation point (say on a similar floor plan where the three blue subgraphs with one articulation point and one dead-end did not exist), each of the two parallel nodes below it would be classified as a CC between two $\mathsf{APChains}$ (line 7).[32] Ultimately, this would result in merging the two chains into one large chain (with a local choice of paths in between). However, none of the two lower chains in Fig. 57 are merged with the right hand chain, because there is also a connected subgraph between all three chains (the condition on line 7).

[32] Neither is an articulation point, because one can go through the other one as well.

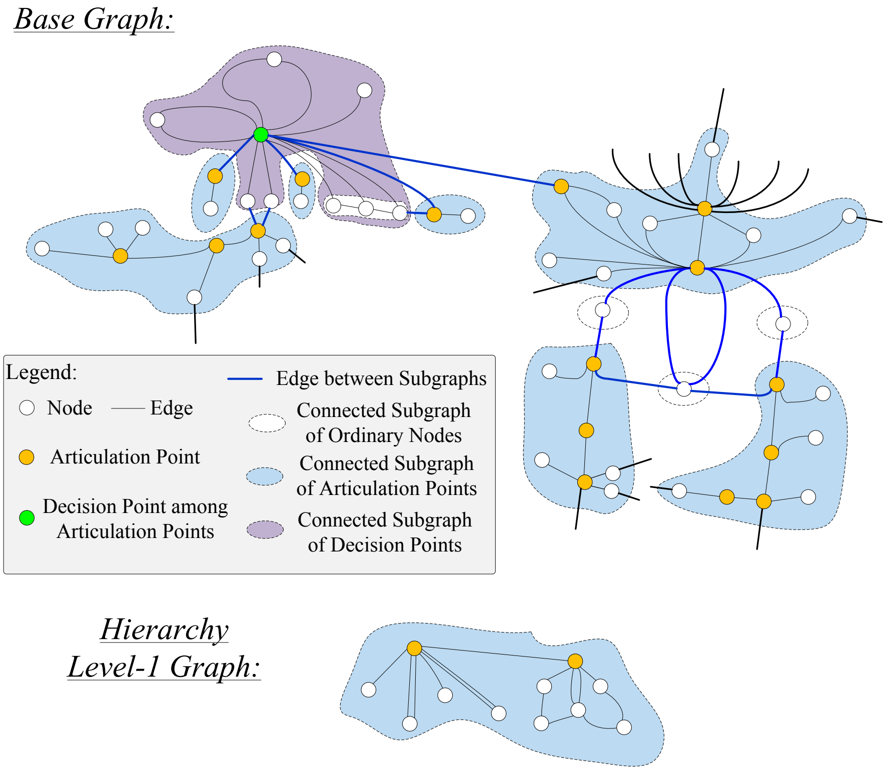

Base Graph:
Legend:
Node
Edge
Articulation Point
Decision Point among Articulation Points
Edge between Subgraphs
Connected Subgraph of Ordinary Nodes
Connected Subgraph of Articulation Points
Connected Subgraph of Decision Points
Hierarchy Level-1 Graph:

Figure 57: Hierarchisation via Articulation Points

Algorithm 4: Hierarchisation of a Dual Region Graph via Articulation Points

```
Input: A dual region graph $G = (N, E)$ from a floor plan.
Output: A coarser hierarchical graph $H = (S, B)$ with subgraphs
        $s \in S$ of $G$ as nodes and edges $b \in B$ between subgraphs
        according to the partitioning of $G$ via articulation points.
find all articulation points (AP) of $G$;
form subgraphs of connected articulation points;
form subgraphs of connected components of ordinary nodes
  (CC);
split the connected articulation point subgraphs into decision
  points (DP) and chains of ordinary articulation points
  ($APChain$);
merge all subgraphs CC which are connected to exactly one
  decision point DP with the subgraph of the decision point;
merge all CC which are connected to exactly one chain $APChain$
  with the subgraph of the chain;
for all CC which are connected to exactly two chains $APChain1$
  and $APChain2$, merge the three subgraphs into one subgraph
  $APChain12$, unless there is another CC subgraph which is
  connected to $APChain1$, $APChain2$, and another chain ;
create a new graph $H$, with a node $s$ for every resulting subgraph,
  and an edge $b$ for every boundary-crossing edge between two
  different subgraphs;
return $H$;
```

Multi-Level Hierarchisation

For larger and more complicated floor plans, it makes sense to apply this algorithm *recursively*, i.e. using the output of the first application as input for the next application etc. (cf. the hierarchy level 1 graph in Fig. 57). This methods yields successive hierarchy levels until the original graph is condensed to a single node. Decision points in the original graph can – but need not necessarily – become articulation points in the next coarser graph. The same holds for connected components between three or more chains.

Number of Levels

Depending on the complexity of the floor plans, there can be more or fewer levels in the hierarchy. The general idea of the presented algorithm is to converge quite quickly, so that there is not an overhead of too many hierarchy levels which represent only minor changes. For example, it makes no difference whether a long corridor is represented as one region or segmented into several regions by intermediate doors (e.g. because of fire security measures). Both representations result in one chain of articulation points.

Discovering the Main Circulation Structure with the Algorithm

A hierarchisation for an entire floor is shown in Fig. 58. For brevity, only the main navigational structure is indicated by different colors: the yellow regions are part of chains, green regions indicate decision points, blue regions dead-ends or connected components and red regions connected components between several chains or decision points. For more details on this and other examples, see Chap. 6 with a comprehensive evaluation.

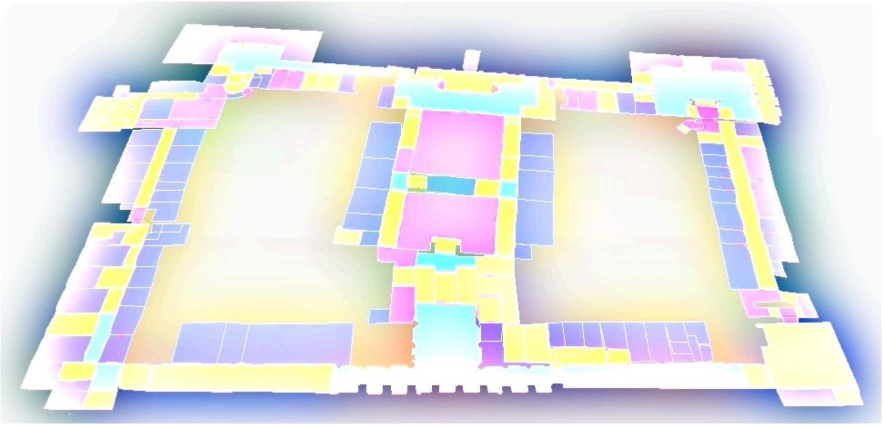


Figure 58: Hierarchisation of an Entire Floor

Animation of a Run

For the presented algorithm a proof-of-concept implementation has been successfully carried out. The implementation has been tested on several floor plans of real buildings. As explained above, the results of this study are shown in the next chapter. However we want to make clear how the algorithm works on a more detailed example, showing a prototypical run.

To give a first impression of this evaluation and to get a feeling for the application of the algorithm (with its recursive behaviour), consider Fig. 59. It illustrates an excerpt from an exemplary floor plan together with the individual steps of the algorithm when the latter is applied to this floor plan. The diagram is to be read from left to right and top to bottom:

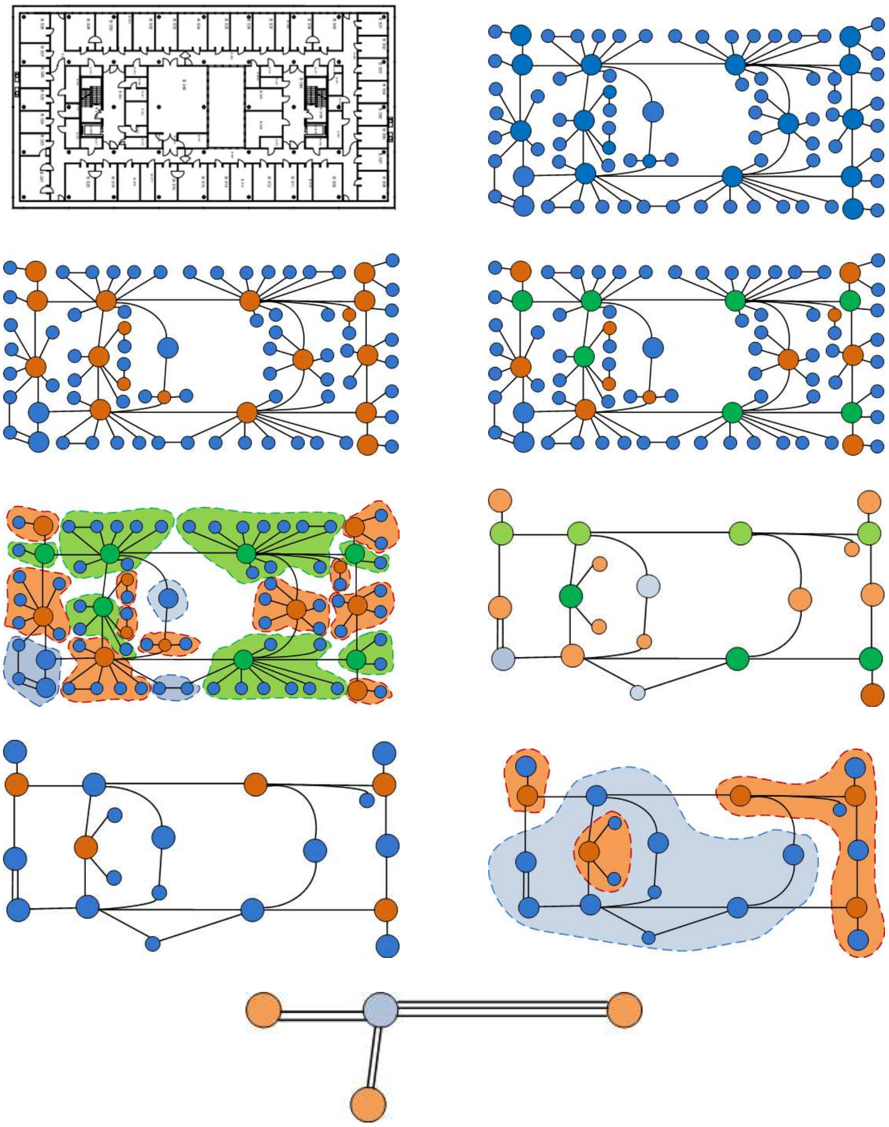


Figure 59: Multi-Level Decomposition of a Floor Plan

Illustrating the Recursive Behaviour

The coloring scheme used in Fig. 59 is the same as for the previous example of Fig. 57: orange nodes indicate articulation points and green nodes indicate decision points which are defined in this context as a special subset of articulation points (with three or more articulation points as direct neighbours). The blue nodes represent 'normal' nodes and those of connected components between articulation points. The first row in Fig. 59 shows the floor plan together with the corresponding graph structure. Articulation points (orange) and decision points (green) are then determined and the corresponding subgraphs are formed (left hand side of third row). Then the original graph is simplified to a coarser graph which models the connections among subgraphs (right hand side of third row). The coarser graph can be obtained by simply replacing each subgraph by a representative node. In this new graph, the process is repeated: articulation points are determined (left hand side of fourth row) and decision points (if there are any). Again subgraphs are formed and the simplification to a coarser graph is applied. This recursive process terminates when there is only one subgraph left, i.e. the next simplification would yield exactly one node.

Factors Influencing the Hierarchisation

There are of course possibilities to tune the parameters of the algorithm. This has an influence on how many hierarchy levels come out: One could for example leave the subgraphs of connected components and not merge them with subgraphs of articulation points. However, this does not seem to be practical since the subgraphs then become very small. Omitting the distinction between decision points and normal articulation points is another option. The result is that the hierarchy often converges quickly with only one level. A complicated circulation system is thus represented as one connected subgraph of articulation points, without the information on decision points. By giving stairs and exits of the building the same status as articulation points (which has been applied in the earlier version of the algorithm [SSO08]), one obtains more decision points. Consequently the hierarchy converges a little more slowly without any additional gain for path finding on the particular floor.

Further Use

Articulation points also offer advantages in combination with context information: For example, one can immediately see which parts of a building are not accessible by wheelchair. One just has to check if all articulation points fulfill the related constraints (e.g. are not of type *stairs*). In old buildings there can be long corridor systems separated by small stairs. In combination with geometric criteria (such as straight line of sight, stretched shape of regions, etc.), one can interpret chains of articulation points as corridors.

Dynamic Changes in Floor Plans

Although the proposed algorithm is quite efficient, it is intended to be used in as a pre-processing step when the hierarchical graph model is constructed. In this regard another question naturally comes up: what about dynamic/incremental changes in the floor plan? This is for example relevant when we want to model a building under construction and provide navigation for it. In Sect. 4.2 we have established a notion of consistency for the individual operations for constructing a hierarchical graph. Since the hierarchisation method employed in this algorithm depends on the structure of the floor plan graph, it is naturally vulnerable to changes in the underlying graph structure.

Possible Solutions

A naive solution would be to run the algorithm again and again each time there is a change in the floor plan graph. If there are frequent changes and updates to the graph structure, this solution is not feasible. A better idea is to store the knowledge on articulation points and decision points used in the first run of the algorithm and check, after each change in the floor plan, whether these assumptions still hold. Note that there could also be new articulation points when an edge is added or removed, and this change could also affect e.g. that former articulation points become decision points etc. In other words, cascading updates are likely so that a compromise seems sensible which consists of bundling changes/updates within a certain period of time (day/week). This problem is interesting from both theory and practice and certainly deserves further investigation in future work.

5.4.4 *Summary*

This section explained the necessity of enhancing the basic geometric model which has been introduced in the previous section to meet practical requirements and relevant cases which can occur in real floor plans. Especially the simple one-to-one mapping to a graph model turned out to be insufficient:

Sect. 5.4.1 presented a handful of examples from real floor plans which cannot be properly handled with a straightforward mapping to a graph structure. The sketched examples are not extremely rare or exotic, but relevant in practice. A systematic analysis of these cases has been carried out. The analysis revealed two main shortcomings of the simple mapping method: on the one hand, there are cases in which the mapping is too coarse (e.g. non-convex polygons). On the other hand, there are cases where an intermediate abstraction level is missing (such as the recognition of a bunch of connected regions being a central/peripheral corridor structure).

In Sect. 5.4.2, a solution was proposed for handling the case of non-convex polygons by a specific way of exact cell decomposition. Unlike standard approaches from Computational Geometry, the proposed solution is *not* based on triangulation. Rather, the method for performing the decomposition takes into account what humans would perceive as 'intuitive' decomposition. The method adheres to principles of Gestalt theory. The main idea is to derive meaningful spaces by the division, and also the division lines themselves take up the role of soft boundaries. Thus, the approach fits nicely into the hierarchical model.

Sect. 5.4.3 covered the second aspect. An algorithm was presented which can extract meaningful subgraphs (which roughly match to concepts such as corridors and wings) from a flat floor plan graph. The algorithm has been successfully implemented and beyond, there is also a proof of concept on several floor plans which justifies the soundness of the underlying assumptions. The idea behind the algorithm is to use the topological structure of a floor plan for navigation: since floor plans are rather sparse, their corresponding graphs can be decomposed around articulation points. Not only does this second decomposition imply a wayfinding strategy or heuristic, but it also promotes the use of the resulting concepts (corridors, wings, etc.) as landmarks, serving as references in route descriptions.

5.5 APPLICATION: QUERY PROCESSING USING THE HIERARCHY

Synopsis

Having defined different means for hierarchisation, the possible employment of the resulting hierarchical graph structure for practical navigation tasks in buildings is now demonstrated. Apart from the inherent advantages of a hierarchy from the point of view of a system's design, such as flexible integration of different networks, it is also shown that common wayfinding issues can be handled with the hierarchy:

PATH FINDING is the first aspect. It can be improved with an underlying hierarchisation (see Sect. 5.5.1). Despite the rather small size of indoor graphs, performance can be an issue if many path queries are posed, or queries involve a class of possible destinations (e.g. *any* lecture room or emergency exit) rather than one single destination. The conditions under which one can exploit hierarchic graph structures (as opposed to flat graph structures) are examined for this purpose.

ROUTE DESCRIPTIONS. The second aspect concerns route descriptions and their generation from the hierarchical model, including the regions' geometry. The basic methods for this are explained in Sect. 5.5.2. Since dual region graphs are abstract insofar as they describe regions as a whole, and not motion through a region, a hierarchisation into convex sub-regions has been proposed. Together with the geometries of the regions, the possible use of this hierarchisation for deriving meaningful route descriptions is shown. Moreover, the coarser hierarchisations into building parts, sections, floors, etc. serve as a convenient means for structuring several route instructions (this process is called chunking).

5.5.1 *Using the Hierarchical Structure for Path Finding*

5.5.1.1 *Overview of Supported Queries*

In general, a complete navigation system to be used in practice should support the following types of user queries [GNSW06]:

LOCATOR – Where am I? Where is object X? (generally, a Point Of Interest) And, especially for pedestrian navigation: What is my *orientation*?

PROXIMITY – Which persons/objects/Points Of Interest are close to my current location? This notion also includes range queries: Which ones are on the same floor/zone or within 50m?

WAYFINDING – How do I get from A to B the fastest/shortest way?

The first point refers to positioning technologies. This aspect is not covered in this thesis, but the locations and current positions of persons, objects, and/or POIs can be annotated as attributes in the graph structure, ensuring independence from a particular positioning technology (cf. Ohlbach et al. [ORLS06] for an overview on diverse indoor positioning systems and technology).

Within the proposed graph model, the term 'near' is understood in the strict sense of walking distance, taking into account the connections between regions via openings such as doors. An indoor positioning system, such as W-LAN fingerprinting [ORLS06], can be used to determine the current location of a user. This location is represented as a point in the geometric model (if the spatial reference systems are correctly mapped to each other). For assigning points of interests to regions (e.g. stating that they are of type *lecture hall* or that a friend/professor is currently in this room), the corresponding nodes in the dual region graph can be annotated. Note that the semantics of distance can be more elaborate with constraints, such as access restrictions. They can be specified in a cost function, which is passed on in the hierarchical graph system as a parameter to path finding algorithms. Hence the hierarchical graph system serves as a flexible framework for adapting general path finding methods to the view and preferences of individual users.

The Role of Orientation

However, one question is eminently important for pedestrian indoor navigation. It is the question '*What is my orientation*?': The egocentric orientation of a wayfinder plays a crucial role in route descriptions, because all landmarks on the way are usually described from this perspective using qualitative spatial relations (left/right of the wayfinder's current position). Thus the spatial reference system of a wayfinder is built around her/his orientation.

What are viable solutions to cope with this issue? One the one hand, it is conceivable that spatial orientations of wayfinders are provided by the currently employed indoor positioning system. Unfortunately, many of these systems are not (yet) capable of determining a user's orientation because this is a technically sophisticated matter. Devices like gyroscopes or accelerometers of high precision have to be employed for this purpose, coupled with other positioning technology to achieve satisfying results in accuracy.

Therefore, for our system design, we cannot rely only on an underlying positioning technology alone to provide us with information on spatial orientation. If information is available from a positioning system, we can gladly use it. But we need to consider as well the case when we do not have any additional information on orientation a priori. Apart from an interface to accept spatial orientations from positioning systems, we need to make some basic assumptions in the absence of this information. These assumptions should be kept as minimal and unintrusive as possible. For this purpose we integrate the concept of spatial orientation in our geometric model and use it for deriving meaningful descriptions for regions. For example, when going from one region to another one, an assumption is made that the orientation at the moment of entry is perpendicular to the traversed portal/door. A methodology for deriving route descriptions which employs spatial knowledge about such implicit orientations is explained in Sect. 5.5.2 in more depth.

5.5.1.2 *Path Finding Example with the Hierarchy*

What is the concrete gain of the hierarchy in terms of path finding? To investigate this issue more closely, consider Fig. 60: It shows a complex building with two floors.

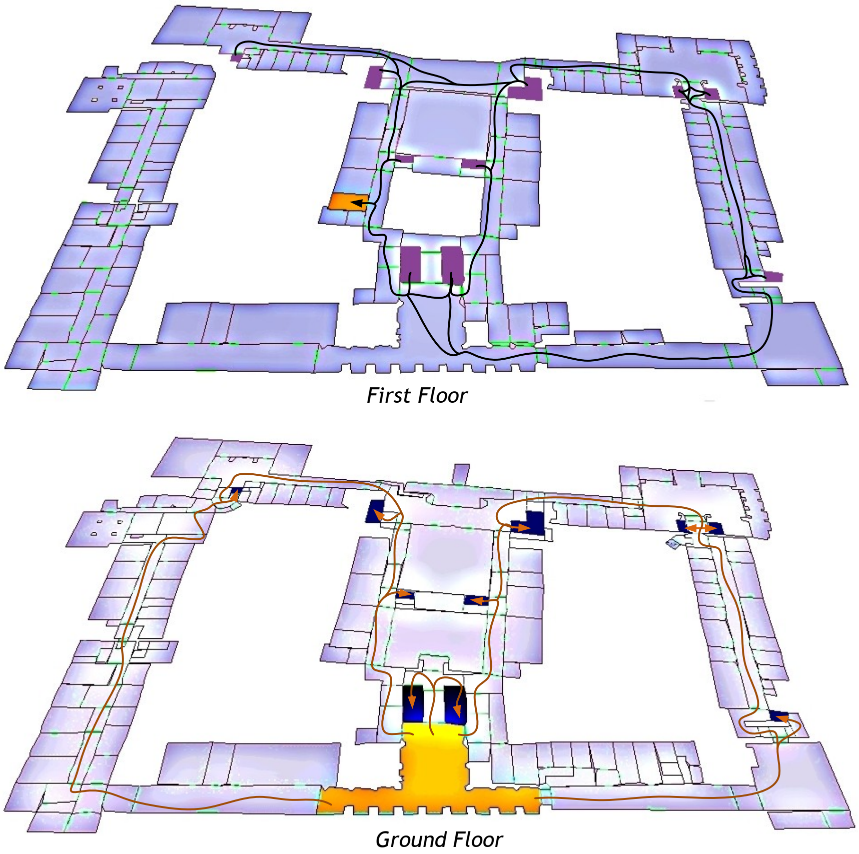


Figure 60: Finding Paths over Two Floors

A prototypical path finding problem over two floors is illustrated in the example. Suppose the task is to find an optimal path from the entrance hall on the ground floor to an office on the first floor (both are marked yellow). One can solve this problem the traditional way by performing a shortest path algorithm on the entire graph spanning across the floors. However, each floor may be stored separately, so that path finding methods are only available on the graphs of floor plans. One can still argue that the graphs of floors can be combined into one large graph. Although this works in the example, constructing one large graph does not scale in general. Say we have two building complexes instead of floors. This makes it difficult to represent the complete unified graph in memory. Besides, buildings from different authorities/owners may not always be combined for reasons of data protection.

However, we can think about a distributed approach with hierarchical graphs: Each floor is represented as a node in an abstract graph (which represents the building), with edges as connections between floors. Each floor has a more detailed graph. Note that the start and end nodes are generally in different graphs when multi-level hierarchies of graphs are considered. So the first task is to find out which graphs contain the start/end nodes. Instead of searching *all* graphs in the hierarchy for contained nodes, it is feasible to use an index structure like a hash table which maps node identifiers to the corresponding graphs. Then one can find out the graph which is the least common ancestor of the graphs containing start and end nodes. In the example of Fig. 60, it is the graph representing the whole building.

Knowing that the two entrance halls and the office are on different floors, one has to traverse some border nodes between the graphs representing the two floors. This is reflected in the hierarchical graph structure – on the coarsest level (the graph representing the whole building), there are ten parallel edges between the nodes 'ground floor' and 'first floor'. Hence there are ten pairs of corresponding border nodes (marked dark blue and magenta) in the dual region graphs of the respective floor plans.

At first glance it seems as if the hierarchy could not provide much of a gain here: computing *all* ten alternatives for connecting the border nodes with the origin/destination on each floor is unfeasible in practice, given that there are a substantial number of common sub-paths. In this case, choosing the traditional modelling could pay off, that is representing the graphs of different floor plans in one large graph so that paths can be determined by a standard algorithm. The underlying problem with the hierarchy is that a single-source single-destination path problem can become a single-source *multiple-destination* problem (between origin and border nodes) combined with a *multiple-source* single-destination problem (between border nodes and destination). In general, there are not too many connections between different floors, although large buildings like the one in the current example can already be problematic. Hence, a *naive* hierarchisation into floor levels alone is inappropriate in such cases.

The previous sections have shown that the hierarchisation of a building is more elaborate – it usually consists of more than just one level within a floor plan. To *avoid* the repeated computation of common subpaths between different boundary nodes, we have to further subdivide the graph structure of a floor. Algorithm 4, which has been presented in Sect. 5.4.3.2, can be employed for this purpose: Fig 61 shows the next hierarchy level for the example as provided by Algorithm 4.

One can perform abstract path finding in the graph below the floor level (shown on the left hand side of Fig. 61). A simple cost function is to give each abstract edge a weight of one, simply counting the traversed nodes S_i. In this case, the abstract shortest path goes from

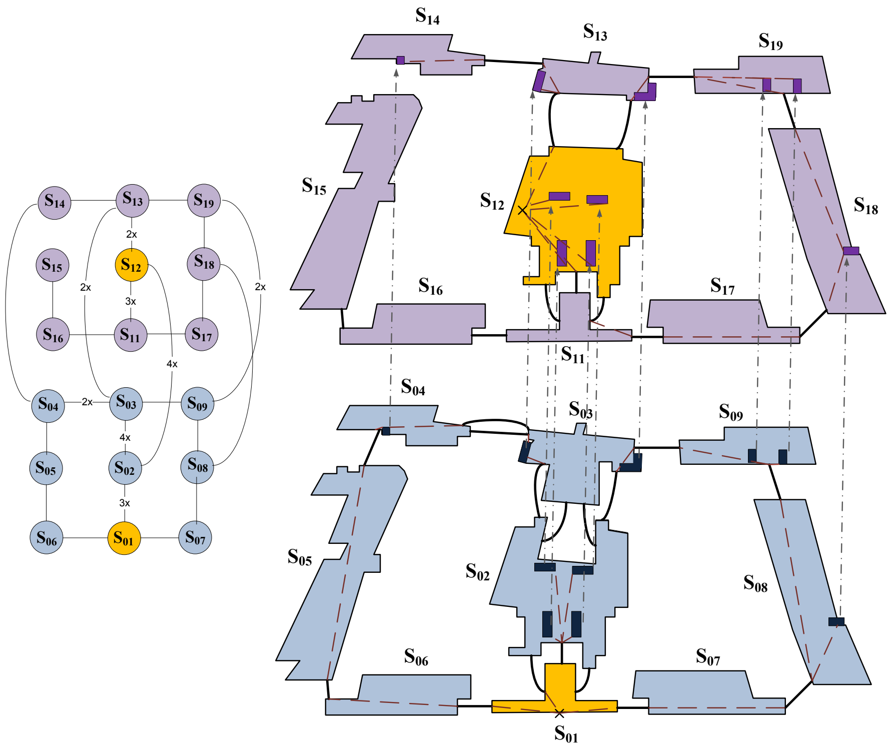


Figure 61: Using a Two-Level Hierarchisation for Path Finding

S_{01} over S_{02} to S_{12} (through either of the three parallel edges between S_{01} and S_{02} and either of the four parallel edges between S_{02} and S_{12}). This abstract path corresponds to 12 concrete paths. If there is another hierarchy level, it can be used in the same manner to assess which of these combinations is preferable.

Refining Paths

The other abstract paths over S_{03}, S_{04}, S_{08}, and S_{09} for reaching the first floor are not further considered at this level because they traverse more nodes. In general, this may lead to sub-optimal solutions since the cost function in the abstract graph is different from the geometric distance. However, there is a simple way for determining whether a more optimal solution through a longer abstract path is possible at all: one can assign for each traversal of an abstract node an estimate of the Euclidean distance between its two border nodes. In Fig. 61, these estimates are illustrated as brown dashed lines across different S_i. If the total length of a path at a more detailed level (such as the shortest refined version of S_{01}-S_{02}-S_{12}) exceeds the estimated length of an alternative abstract path (e.g. S_{01}-S_{02}-S_{03}-S_{13}-S_{12}), then the alternative abstract path is a promising candidate worth being refined. In the next hierarchic level, the corresponding shortest path algorithm verifies whether the detour over S_{03} and S_{13} is indeed shorter (as the initial estimation suggested). Then the estimate is corrected with the new information gained from the finer hierarchy level.

Admissible Heuristic

It is worth noting that this method of estimating the lengths of abstract paths is 'safe' insofar as it can only *under*estimate the true distance (due to the properties of Euclidean distance), but never overestimate it. Thus it is clearly an admissible heuristic, just as it is used in the $A^{\star}$ algorithm [HNR68]. However, notice that this algorithm is slightly different from the hierarchical $A^{\star}$ algorithm discussed in Sect. 3.3: whereas hierarchical $A^{\star}$ performs on a flat graph (and uses the hierarchy *only* in the heuristic itself), the approach of using a heuristic here updates the cost functions in a true hierarchical graph system (according to the heuristic estimate).

This example showed the subtleties and explored different options for exploiting a hierarchy for path finding. In addition to the three classic variants for hierarchical path finding presented in Sect. 3.3, new ideas of using the hierarchy were brought in. Unfortunately, within the limited time available for this work, it was not possible to implement a path finding algorithm which makes use of all the ideas mentioned above. Further research is necessary in this respect. For the time being, we have stuck to a classic variant of hierarchical optimisation with materialisation of costs: a simple version of the algorithm presented by Jing et al. [JHR98] has been prototypically implemented. It is optimal and makes heavy use of precomputed shortest paths. Nonetheless, it would be interesting to study the differences between several hierarchical path finding algorithms and analyse under which conditions one performs better than the other (and why). This topic remains for further studies building on this work.

5.5.2 *Principles and Versatile Methods for Deriving Route Descriptions*

Guiding pedestrians in buildings is a challenging problem. How can route descriptions be derived from a hierarchical indoor model, ideally in an automatic way? For this purpose, the underlying principles of giving route descriptions are first elucidated from a theoretical point of view. Then – based on these principles – versatile methods are presented for obtaining route descriptions from a floor plan's geometry and its subsequent hierarchisation. Parts of this section were presented by the author in a research paper at the AGILE conference [OS08].

5.5.2.1 *From an Allocentric to an Egocentric Perspective*

Shifting Perspectives: Following Routes

It is assumed that the basic hierarchical representation of the environment has been precomputed from the available geometric data of a building's blueprints according to the methods presented in Sect. 5.3 and 5.3.4. This hierarchical model is initially **allocentric**: The perspective is not centred on an individual navigating in the environment, but rather on the environment itself (i.e. it is top-down, from a bird's-eye view). Information represented in such a model is complete and can, in particular, be used at the stage of route planning (as explained before in Sect. 5.5.1). Once a suitable path has been determined through the building, we can switch perspectives; we can take on that of the wayfinder, who tries to *follow* the suggested route.

In this **egocentric perspective**, motion through every region on the route has to be represented and described. The local structure of each region has to be considered to this end, with the particular entry and exit points which are fixed by the given path. They are represented as openings on the region's boundaries. Non-convex regions and regions with inner obstacles require special treatment: Navigating these regions can be too complicated without precise instructions to follow. Additional information, e.g. in form of intermediate waypoints, may be required for such regions with a complicated structure. The implicit openings determined by the geometric decomposition method described in Sect. 5.4.2 qualify as suitable intermediate waypoints. In short, factors which have an influence on the complexity of giving instructions are:

- the number and configuration of openings on a spatial region's boundary (e.g. some could be close to each other or on the same boundary while others are more distant),
- the number and configuration of non-convex corners (with limited advance visibility around them). Possibly, intermediate decision points *within* the region are implied,
- the configuration of several openings pertaining to adjacent regions (e.g. if they are in line or at different angles).

5.5.2.2 *Preliminaries*

The aim of this work is *not* to derive route descriptions which are sophisticated from a linguistic point of view (see Sect. 7.2.2 for different models). Natural Language Generation is a complex topic beyond the scope of this thesis. Instead, complementary to these methods, general principles are discussed and versatile methods are elaborated for deriving abstract descriptions from the underlying geometry (by considering the spatial relations between openings). Note that the following considerations are also helpful for the design of a linguistic component which creates human-friendly instructions to be used in the human-computer interface of an indoor navigation system.

Principles

A feasible approach for measuring the pragmatic information content of route descriptions is to adopt an information-theoretic viewpoint. This is explained in Frank [Fra03]: descriptions are considered to be equivalent if they lead to identical actions. However, we do not want to compare two different descriptions here, but to minimise the ambiguity and imprecision of an individual description. This can be measured with entropy too (in the sense of assigning probabilities for spatial decisions):

Pragmatic Information Content of a Description

It is assumed that a wayfinder is unfamiliar with the environment. Consequently, without being given descriptions how to traverse regions, the wayfinder has to make *uninformed* decisions (unless there are some cues built into the environment, such as signs). What does the situation look like for a wayfinder in a particular region R? Each opening e_0 on the boundary of R constitutes a possibility for traversing the region in a different way (and continue through another region). Thus a certain probability $p(e_o)$ that the opening e_o is chosen by the wayfinder can be assigned to each e_o. As noticed by Turner and Penn [TP02] and Richter et al. [RK04, HKR06], it seems as though these probabilities are not evenly distributed and that there is a bias to continue in the same direction (moving straight on). Still, unspecific instructions such as *'take the door straight ahead'*, or *'go straight on/through the hall'* leave room for interpretation, especially in constellations where several doors are right in front of the wayfinder (see Fig. 63). Apart from this exception, one only needs to describe motion through a region *explicitly* if a change of direction occurs. The method to compress several instructions of the type *'go straight ahead'* into one instruction (*'continue straight ahead until ..'*) is called line-of-sight chunking [RK04] or, more generally, spatial chunking.

Reducing Choices with Spatial Knowledge

The **spatial knowledge** communicated in an instruction, telling e.g. whether the next door on the path is *to the left*, *to the right*, or *in front* of the wayfinder, helps to reduce the number of choices for a wayfinder. If the instruction is indeed precise and unambiguous, there is only one choice implied by it. In practice it is often necessary to count the openings on the same boundary in a particular order, e.g. for instructions like 'take the *second* door to your left'. Thus one has to determine what people perceive as one coherent boundary (including several openings) from a planar floor plan graph. This is described in Alg. 5:

Determining Coherent Boundaries

The algorithm starts from a corner C_i in the polygon which represents the boundary of the region R and proceeds in counter-clockwise direction (i.e. from i to $(i+1) \mod n$ etc.). Although the list of corners is not cyclic, the traversal is cyclic due to the modulo operation. If the internal angle $\alpha(C_i)$ of the corner C_i enclosed in the polygon (see Def. 5.4.1 in Sect. 5.4.2) is around 180°, i.e. $\alpha(C_i) = 180° \pm \delta$ for a sufficiently small value δ (e.g. $< 30°$), then the two boundary segments $\overline{C_{i-1}C_i}$ and $\overline{C_iC_{i+1}}$ are put together into one coherent boundary $b = C_{i-1}C_iC_{i+1}$ and the algorithm continues in the same way with C_{i+1} and so on. This way further boundary segments are collected into a coherent boundary b until, eventually, a corner C_{i+n} is encountered with an interior angle $\alpha(C_{i+n})$ above or below the threshold. Note that boundary segments $\overline{C_iC_{i+1}}$ correspond to edges in the planar floor plan graph. Thus they can also be openings. Since the starting point C_i can be in the middle of the coherent boundary b, one needs to apply this method also from C_i in reverse direction, i.e. clockwise, to collect further boundary segments which belong to b. When this method is applied analogously to all remaining corners of the polygon which are not in b, one obtains all coherent boundaries of a polygon. Coherent boundaries b are ordered counter-clockwise, so all openings on them can be counted fairly easily.

Example

Consider the concrete example illustrated in Fig. 62:

In the example, openings are represented as line segments B_i and ordinary boundaries as line segments L_j. Four angles are above the threshold of δ, namely the angles between the line segments L_3 and

Algorithm 5: Determining the Coherent Boundaries of a Region

```
Input: A polygon $P = C_1 C_2 \dots C_n$ (a non-cyclic list of corners)
       representing a region in a planar floor plan graph.
Output: A set $\mathcal{B}$ of connected line segments $b$ which are
        perceived as coherent boundaries.
$\delta := 30°$;
mark all corners $C_1 \dots C_n$ as unfinished;
while there are unfinished corners $C_i$ do
    choose one corner $C_i$ which is unfinished;
    create a new, empty line segment $b$;
    add $C_{(i-1) \bmod n} C_i$ to $b$;
    mark $C_{(i-1) \bmod n}$ and $C_i$ as finished;
    $j := i \mod n$;
    while for the internal angle $\alpha(C_j)$: $|180° - \alpha(C_j)| < \delta$ do
        add $C_j C_{(j+1) \bmod n}$ to the tail of $b$ ;
        mark $C_j$ as finished ;
        $j := (j+1) \mod n$; // proceed counter-clockwise and
        cyclic
        if $j == i \mod n$ then break;
    end
    $j := (i-1) \mod n$ ;
    while for the internal angle $\alpha(C_j)$: $|180° - \alpha(C_j)| < \delta$ do
        add $C_{(j-1) \bmod n} C_j$ to the head of $b$;
        mark $C_j$ as finished;
        $j := (j-1) \mod n$;  // proceed clockwise and cyclic
        if $j == (i-1) \mod n$ then break;
    end
    add the line segment $b$ to $\mathcal{B}$;
end
return $\mathcal{B}$;
```

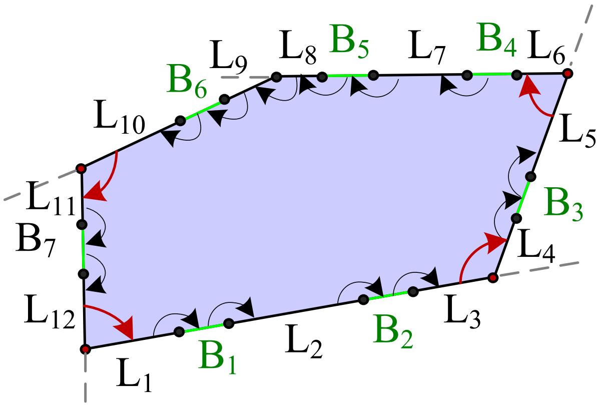


Figure 62: Determining the Coherent Boundaries of a Region

L_4, L_5 and L_6, L_{10} and L_{11}, and finally between L_{12} and L_1. In contrast, the angle between L_8 and L_9 is within the range. So are all angles between B_i and L_j. As a result, there are four coherent boundaries. They are separated by the corners marked in red.

Introducing B_{e_o} *and* $\Omega(B_{e_o})$

Openings apparently play a major role for giving route directions. They serve as waypoints on a path and are referred to in directions. To better describe the spatial relations between different openings of a region, for each opening e_o a **waypoint** B_{e_o} is assigned which is the geometric centre point of the edge e_o in the planar floor plan graph. Additionally, a **reference orientation line** $\Omega(B_{e_o})$ is assigned which goes through B_{e_o} and represents the intrinsic orientation of the opening. By default, it is perpendicular to the edge e_o. The background to this is that the orientation of the wayfinder – at the moment of entry through the opening e_o – is represented by $\Omega(B_{e_o})$. This spatial knowledge is relevant for giving subsequent instructions within the region, as detailed in the next sections.

5.5.2.3 *Deriving Descriptions for Convex Regions*

To begin with, let us have a look at convex regions because they are generally easier to deal with. Say a routing algorithm has provided a pair of openings e_e and e_l through which a region R is entered, respectively left. An example is shown in Fig. 63:

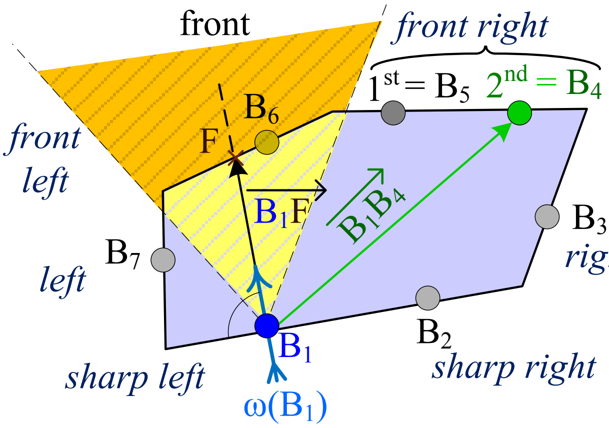


Figure 63: Egocentric Directions

Example

Here, the openings on the region's boundary are represented as waypoints, numbered in counter-clockwise order from $B_1 = B_{e_e}$ where the region is entered. We want to obtain a description of the path from $B_1 = B_{e_e}$ to $B_4 = B_{e_l}$. How can this be accomplished? The key idea for deriving descriptions is to partition the openings on the region's boundary into those which are to the left and those which are to the right of the wayfinder's orientation $\Omega(B_{e_e})$ when entering the region via the waypoint B_{e_e} (see Fig. 63). In addition, it is useful to determine whether the opening e_l is in front or to the back of e_e. The complete algorithm for finding out the relation between e_e and e_l with a path through R is explained in Alg. 6:

Algorithm 6: Deriving Route Descriptions for Convex Regions

```
Input: A convex region $R$, two openings $e_e$, $e_l$ on the region's
       boundary, and the set $\mathcal{B}$ of coherent boundaries of $R$.
Output: A textual route description descr for a path from $e_e$ to
        $e_l$ through $R$.
extend $\Omega(B_{e_e})$ through $B_{e_e}$ and determine the intersection point
$F$ with the region's boundary ;
if the waypoint $B_{e_l}$ is to the left of $\overrightarrow{B_{e_e}F}$ then direction := left;
else direction := right;
heading := undefined;
if $B_{e_l}$ is on the same coherent boundary $b_F$ as $F$ and $b_F \neq b_{e_e}$ then
    heading := front;
    count := number of openings of the same type as $e_l$ on the
    coherent boundary $b_F$ between $F$ and $B_{e_l}$;
    descr := "take the <count + 1> <$B_{e_l}$.type> to the front
    <direction>";
end
if $B_{e_l}$ is on the same coherent boundary $b_{e_e}$ as $B_{e_e}$ and $b_F \neq b_{e_e}$ then
    heading := back;
    count := number of openings of the same type as $e_l$ on the
    coherent boundary $b_{e_e}$ between $B_{e_e}$ and $B_{e_l}$;
    descr := "make a U-turn: turn <direction> and take the
    <count + 1> <$B_{e_l}$.type> to your <direction>";
end
if heading == undefined then
    if $b_F == b_{e_e}$ then // circular room with one smooth
    boundary
        countF := number of openings of the same type as $e_l$ on
        $b_F$ between $F$ and $B_{e_l}$;
        countE := number of openings of the same type as $e_l$ on
        $b_{e_e}$ between $B_{e_e}$ and $B_{e_l}$;
        if countE $\leqslant$ countF then ; // nearer to $B_{e_e}$ than to F

        descr := "take the <countE + 1> <$B_{e_l}$.type> to your
        <direction>";
        else descr := "take the <countF + 1> <$B_{e_l}$.type> to the
        front <direction>";
    end
    else
        descr := "turn <direction>";
        apply the algorithm recursively on the polygon defined by
        $B_{e_e}$, $F$, and all points which are to the left/right of $\overrightarrow{B_{e_e}F}$
        (according to the value of direction), replacing $e_e$ by $\overline{B_{e_e}F}$
        as input value;
        concatenate the resulting description to descr;
    end
end
return descr;
```

Algorithm Applied to the Example

In the concrete example of Fig. 63, a line is created through B_{e_e} in the direction $\Omega(B_{e_e})$ and it is extended. The intersection point F of this line with the region's boundaries is determined. Then, a simple left test is performed for all openings w.r.t. $\overrightarrow{B_{e_e}F}$ (lines 2,3): the openings are sorted according to which side they lie. Thus, B_2 to B_6 are all to the right (including $B_{e_l} = B_4$), while only B_7 is to the left of this line. The next step of the algorithm is to test whether B_4 lies straight ahead. Therefore the coherent boundary which contains F is considered (lines 5-9). In the example, B_6, B_5, and B_4 are on this boundary (in this order from F). The opening B_4 is the third to the front right. However, it would not be described this way, since B_6 is very close to F. B_4 is the second opening if B_6 is not counted. It is also distinguished from B_3, which is to the right of F, but not on the same boundary. An opening is to the back of $\overrightarrow{B_{e_e}F}$ (lines 10-14) if it happens to be on the same coherent boundary as B_{e_e}. The corresponding manoeuvre is a U-turn, either to the left or to the right. Circular rooms are a special case (lines 16-22), where there is exactly one coherent boundary for the whole region. Depending on the proximity of B_{e_l} to F or B_{e_e} ($\mathtt{countF}$, respectively $\mathtt{countE}$), a suitable instruction is given. A simple left or right turn (lines 23-27), e.g. from B_1 to B_7/B_3 results in a turn action followed by a move straight action. It can be determined by partitioning the left/right hand side of $\overrightarrow{B_{e_e}F}$ again.

Describing Actions in a Formal Language

In a preceding diploma thesis supervised by the author [Sch06], a general-purpose formal route description language, called `PlanML` has been proposed for different kinds of spatial networks. This language has an XML-based syntax for a unique representation of different plans from different network types. A sample for the example of Fig. 63 is shown below in PlanML syntax:

```
<plan>
  <location id="R" type="region">
    <location id="B1" type="opening" />
    <action id="B1B4">
      <data>
        <action-description>
          Take the second door to the front right.
        </action-description>
      </data>
    </action>
    <location id="B4" type="opening" />
  </location>
</plan>
```

However, if we look more closely at the example, it turns out that the language is too unspecific for indoor route descriptions. Additional constructs are needed: Besides the missing conceptualisation of simple turn actions, a notion for counting the number of openings is also required (see Alg. 6), as well as conceptualisations of complex manoeuvres such as U-turns. Another option is to use the cognitive OpenLS specification as proposed by Hansen et al. [HKR06] which is more expressive and builds on a standard. Unfortunately, the integration with that formal route description language could not be achieved within the given time. This topic is left for future work.

5.5.2.4 *Deriving Descriptions for Non-Convex Regions*

Compared to their convex counterparts, non-convex spatial regions[33] are admittedly encountered more rarely in floor plans. Despite this, handling these cases is extremely important in practice: Human wayfinders are likely to get lost in complex regions like large halls or branching corridor systems without guidance.

Using the Decomposition

In Sect. 5.4.2, a geometry-based method has been introduced which explains how to decompose non-convex regions in a specific way into convex parts. The decomposition lines are not arbitrary, but guided by the idea of extending architectural lines and closing incomplete shapes. As a result, a finer-grained graph is obtained. It describes convex sub-regions within the region separated by decomposition lines. This decomposition can be exploited for giving directions through non-convex regions. For an understanding of the basic idea, consider the example illustrated in Fig. 64:

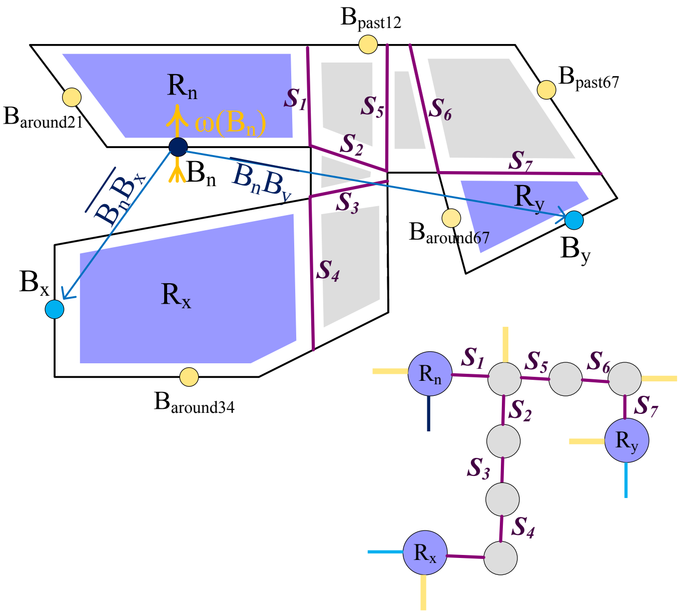


Figure 64: Using the Decomposition

The starting point is a decomposition such as that obtained by the method in Sect. 5.4.2. In the example of Fig. 64, the decomposition lines S_1 to S_6 (represented as purple edges) have been created together with a corresponding fine-grained graph of sub-regions. Supposed the task is to describe the paths through the region from B_n to B_x and B_y (see Fig. 64): First, the fine-grained graph is used to plan an abstract path through the region. The involved sub-regions R_n, R_x, and R_y which are the start/end nodes of this path are marked in blue. According to the number and position of decomposition lines,

[33] including also those regions which have inner obstacles

sub-regions on these paths can be classified into different categories: the sub-region enclosed by S_1, S_2, and S_5, for example has three internal connections. It is a T junction. The other sub-regions have only two or one internal connection(s). The two regions enclosed by S_3, S_4, respectively S_5, S_6 are almost orthogonal connections around corners.

Determining Visibility

We can now assess whether B_x and B_y are visible from B_n *without* checking the intersections of the line of sight $\overrightarrow{B_nB_{x/y}}$ with the region's boundary (i.e. polygon): instead, all implicit boundaries S_i which are edges in the determined path from R_n to R_x/R_y are tested for intersection with $\overrightarrow{B_nB_{x/y}}$ in the order of their appearance on the corresponding paths.[34] In the case of B_x, the order is (S_1,S_2,S_3,S_4), and for B_y it is (S_1,S_5,S_6). Although the line of sight $\overrightarrow{B_nB_y}$ incidentally intersects S_3, one does not need to determine this because the path from R_n to R_y does not lead through the edge S_3. If two openings are mutually visible, then the corresponding path description can be compressed by means of chunking.

Combining the Descriptions of Convex Sub-Regions

Having determined the abstract path through the sub-regions, the basic method for describing paths in convex regions can be applied to each sub-region (considering the decomposition lines S_i as special kind of openings which are not referred to though). These partial descriptions can be combined in the following manner: Care has to be taken for very long coherent boundaries which stretch across adjacent sub-regions – if there were, for example, more doors on the upper boundary in Fig. 5.4.2 besides B_{past12}, the counting of these openings has to be done over the division of sub-regions. For longer 'move forward' sequences such as ($B_{around21}$, S_1, S_5, B_{past56}), likewise, the individual instructions can be replaced by one single *'go straight ahead'* instruction. Optionally, reassuring cues such as *'move past the junction'* (referring to the region enclosed by S_1, S_2, S_5) could be given when the forward movement exceeds certain length. For reaching B_y from B_n the corresponding description would be *'Turn right and go straight ahead'* (obtained from R_n), followed by *'Pass the junction'* (from the region enclosed by S_1, S_2, S_5) and *'Turn right [at the corner]'* (referring to S_5 and S_6). The final instruction, *'Take the door straight ahead'*, is obtained from R_y. In an analogous way, the description from B_n to B_x can be obtained. It contains the chunks *'Turn right [at the junction]'* (S_1, S_2, S_5) and *'Turn right [at the corner]'* (S_3, S_4).

Route Instructions in a Hierarchy

Note that a similar technique can be applied for concatenating descriptions of adjacent regions in a floor plan graph. For improving the quality of these descriptions, it is helpful to refer to the type of region if it is known. Especially transitions between different types of regions (e.g. between a large hall and a corridor, or a room and a corridor) could be incorporated in the descriptions. Thus, one can enrich route descriptions with labels and types of regions. An

[34] Note that this method is usually faster than checking intersections with all boundary lines of the polygon, since there are fewer elements S_i. Moreover, these S_i are already in the correct order, i.e. if the first line S_i is not intersected, the others do not need to be checked any more.

additional frame for structuring several instructions into an abstract, higher-order instruction is provided by the hierarchy levels of floors, sections, wings, etc. such as determined by the other hierarchisation algorithms. Folding and unfolding of abstract instructions is a convenient feature of hierarchically organised instructions. This feature is also an integral, built-in part of the `PlanML` language [Sch06] devised by a diploma student of the author.

5.6 SCENARIOS FOR PEDESTRIAN INDOOR NAVIGATION

In the previous sections the basic principles of a pedestrian indoor navigation system were highlighted and a novel model based on hierarchical graphs has been proposed. The implementation of a complete, ready-to-use navigation system could not be finished within the time of this research. Nonetheless, some prototypes exist for individual components (see Chap. 6 for an overview). These components need to be further developed and assembled into a complete navigation system. The previous sections have made clear which questions are relevant for the design of such a system tailored to people's needs. The proposed algorithms can be used as guidelines for implementing such a system. The following paragraphs give an idea on possible scenarios where such a complete system can be useful. The general importance of pedestrian indoor navigation should become apparent and also the applicability of an automated system for pedestrian guidance in other scenarios.

AIR TRAVEL. Increased numbers of plane movements in connection with ever larger aircraft result in an increased load on an airport's infrastructure. However, this problem cannot always be met by structural expansion alone. Dynamic and efficient guidance of passengers can to some extent counteract these developments. Passengers are often under time pressure since delays and short connections are common nowadays. This is especially the case for international flights, where transits take place at foreign airports. Increased and longer security checks as well as fewer available ground staff for this purpose make all aspects of air travel more demanding.

Automated assistance and guidance of people, e.g. via mobile devices, could substantially help in a number of respects: finding facilities such as information booths, departure gates, baggage claims or parking lots could be assisted by route finding applications. Changes and updates of all kinds (e.g. concerning a certain flight or company) could be communicated via Personal Digital Assistants (PDAs) or mobile phones in an intelligent manner, i.e. only to those people actually affected by the change. Moreover, indoor positioning and dynamic event handling could aid the ground staff in locating passengers who are late for departure. Paper tickets can be replaced by electronic counterparts and check-in can also be done electronically, e.g. using 2D-matrix-code sent via MMS to mobile phones[35] to reduce queues and waiting time.

healthcare. A similar situation can be found in large clinics or hospitals. Modern hospitals often consist of complex infrastructures integrating all kinds of special units, employing a wide range of experts, and implementing complex processes in order to cater for the needs of patients. Patients and visitors are normally neither familiar with the often large complexes, nor with individual procedures. Factors such as time pressure or exceptional stress aggravate problems of spatial orientation within the underlying infrastructure.

Especially today, where human resources are scarce and must be deployed with optimal efficiency, people can benefit from automated assistance. Locating not only certain premises but also employees, who often move about in the building, is a recurring task for both patients and staff. Active badge systems have been devised to eventually replace beepers. Patients' vital signs could be monitored and their movements tracked for safety reasons. People would not be confined to areas with constant audiovisual supervision and thus staff could concentrate on more important matters.

education. University life is dynamic in a number of ways: Students have different assignments and lectures changing more or less each semester. The general facilities are shared by a number of different units and their staff. Changes to and exceptions from schedules occur rather frequently – especially at the end of semester when most examinations take place. Moreover, university campuses are often larger complexes; they consist of a number of different buildings. These buildings are sometimes not even on the same or adjoining premises, but scattered across different parts of a city. Finding certain staff or facilities can therefore be a difficult task, especially for freshmen who are not yet accustomed to this way of living.

Generally speaking, the tasks of managing the curricular and extra-curricular activities, personal schedules, deadlines, and other administrative concerns must all be carried out in a mobile environment. The reason is that students, unlike staff, do not have "offices" of any kind. Even staff have to conduct a considerable part of their business outside of their offices. Web information systems have become a valuable part of the information infrastructure for universities, although few possibilities of truly mobile access exist[36]. In this environment, information systems could regularly send reminders for important deadlines or schedule updates. Say a student wants to contact a professor, then route finding algorithms could include additional information such as time and location to provide a more adequate

[35] http://www.heise.de/newsticker/meldung/70240

[36] While there exists wireless LAN which can be accessed by laptop computers in many places, we are looking towards smaller (therefore *truly* mobile) devices such as PDAs and mobile phones.

answer. Instead of visiting the office during consultation hours for example, it could be easier to postpone the meeting until the professor is lecturing in a room nearby. This would involve knowledge of both the professor's and the student's schedule and the respective locations.

5.7 SUMMARY

This chapter covered a broad range of different themes which come up when a hierarchical graph model is being applied to indoor environments.

Section 5.1 explained the general benefits of a hierarchical representation for indoor environments and motivated them by examples. In Section 5.2, the architecture of a complete system was sketched, illustrating the interdependencies of individual components. However, this complete architecture is more wide-ranging than the implementation which has been carried out in the course of this thesis (see Chap. 6 in the following). Section 5.3 then gives a detailed discussion of the individual steps required to extract a graph from floor plan, and how a hierarchy can be created in this course. Different construction algorithms are introduced for achieving this task. The nesting of spatial regions and problems in connection with the third dimension are also discussed. Hierarchical graphs are proposed to model all these aspects.

As discussed in Section 5.4, there were still many exceptions and cases which could not be handled by the basic system. Floor plans of real buildings were used for a survey and systematic analysis of problematic cases. As a consequence of this, two concrete enhancements were proposed for the basic system: first, a novel geometric decomposition was introduced which produces convex and naturally looking spaces within non-convex spaces. Then, an algorithm was presented which recursively divided a floor plan graph around its articulation points and extracted meaningful subgraphs such as corridors, wings, or sections.

The following section further justified the use of hierarchical graphs by analysing how queries can be processed in the presence of such a hierarchy. First, queries were categorised an it was analysed how they can be answered with the proposed system architecture. The two aspects of path finding and generation of route descriptions were mainly discussed there. A methodology for deriving route descriptions from spatial regions was presented which is makes use of implicit spatial orientations for the derivation of qualitative spatial relations. It can be seen as a first step preceding natural language generation. Finally, three scenarios of pedestrian indoor navigation were illustrated in Section 5.6 to give an impression of the impact of this research in future scenarios.

6 STATUS REVIEW AND EVALUATION

In this chapter an overview is given on the progress of implementation. Some basic components could be realised as first prototypes in the scope of this thesis. As a prominent example, the algorithm detailed in Sect. 5.4.3 was implemented and then tested on several floor plans. The results are presented in Sect. 6.2. With an analysis carried out on several floor plans of real buildings, we have not only gained a first confirmation of the validity of our approach, but we can also make statements about the expressiveness of our model and compare it with other models. Note that a few components still need to be completed. Even though their implementation is not complete, the underlying ideas and principles were presented in this thesis. For example, the process of generating route descriptions was brought up without going into the details of the complex topic of natural language generation. This is left for future work.

6.1 OVERVIEW ON THE IMPLEMENTATION

Core Functionality

First, there are the components which consitute the core functionality of the hierarchical graph system. They have been realised in Java. Among others, loading and storing graphs is possible from an XML-based representation, OWL ontologies, and a common graph drawing format. An open source graph framework has been employed and extended for the basic functionalities of a single graph. Of course, the construction of hierarchical graphs is the most important aspect: Different graphs are linked together in a mediator-based system (see Sect. 4.1.2 for details). The mediator system has been prototypically implemented for two levels of hierarchy. For path finding, a variant of the hierarchical encoded path view [JHR98] has been implemented. It makes use of massive precomputations of paths.

Context Information

Context information is represented by means of ontologies which are linked to nodes and edges of different graphs [Ide06, Ide07]. However, it turned out that the hierarchical encoded path views are not compatible with a flexible evaluation of cost functions. The basic graph algorithms have been generalised insofar as they now accept a set of boolean constraints as parameters. This allows one to instantiate a graph algorithm with a specific set of constraints (telling which node and edge types are excluded from search). The underlying graph does not need to be changed because the behaviour of the algorithm can be altered in a generic way.

Moreover, there are the specific algorithms and data structures for indoor environments: The systematic method presented in Sect. 5.3 has been implemented and applied to several floor plans. The map modelling toolkit Yamamoto [SH06] developed at the university of Saarbrücken served as a basis for creating vector-based geometries of several floor plans. The basic algorithms for graph construction have been applied to these geometries. Two floors of the same building were used for testing the hierarchisation in the third dimension. Two enhancements were proposed in Sect. 5.4. The idea of geometric decompositions in Sect. 5.4.2 has not yet been realised in a suitable algorithm due to time restrictions. The implementation of the other method for extracting meaningful subgraphs from a dual region graph (see Sect. 5.4.3) has been carried out within this thesis. Therefore, first results of an evaluation on floor plans are available. They are presented in the next section. Concerning the methods for applying the hierarchy (see Sect. 5.5), only some basic functionalities have been implemented. Therefore, these methods could not yet be exhaustively tested. This is left to future work.

6.2 EVALUATION OF THE HIERARCHY ON REAL FLOOR PLAN DATA

Accuracy and Expressiveness

With the preceding analysis of real floor plans, we can make some statements about the expressiveness of the proposed hierarchical model. Although we have initially employed the Yamamoto editor for creating the geometric models by means of overlaying an image of a floor plan and drawing polygons, a few shortcomings of this system became apparent: first it has a limited accuracy because images of floor plans are not allowed to be larger than 2048 × 2048 pixels.[1] Thus, for an exemplary building of 200 metres for instance, the maximum accuracy is roughly 10 centimetres per pixel. This may not be sufficient in all cases because the angles in small rooms may not be captured accurately enough. This mismatch could ultimately lead to navigation errors due to wrong angles. Another, less dramatic effect of the maximum resolution per pixel is that walls which are thinner than 10 cm can only be represented as lines (and not as polygons/obstacles).

Furthermore, our geometric data model is more expressive than thus of Yamamoto which only accepts valid polygon models. Our focus was to cover as many practical cases and instances of floor plans as possible in a general model. We have relaxed the conditions of polygons and can process more general, vector-based models. The basic geometric model consists of floor plan graphs (see Sect. 5.3.2.1). This model is quite expressive. For example, there is no way to model thin Spanish walls (see Fig. 32 in Yamamoto and neither inwards-oriented walls[2] due to the combination of coarse maximum resolution and required polygon model. Our approach also derives implicit information from a floor plan graph (as explained in Sect. 5.3.2.2), for example the nesting between different regions. This is not only the case for room-in-room constellations and circular corridors, but also

[1] This limitation is due to the resources used for 3d rendering.

[2] frequently occurring in museums

for any kind of obstacles inside rooms (like furniture). In general, the nesting of regions can be recursive. This aspect has been neglected in the Yamamoto system so far. In contrast, the strength of the Yamamoto system clearly lies in the capabilities and functionality of its graphical display. It has a 2d and 3d rendering engine which even allows one to virtually walk through floor plan models.

Beyond this, the resulting hierarchical graph model is very flexible because it can model a number of different aspects which cannot be captured in equal measure by a flat graph model. The different steps of constructing the hierarchical model for indoor environments have been laid out in Chapter 5. In particular, the hierarchisation algorithm presented in Sect. 5.4.3 has been tested on several floor plan models.

The floor plans that have been considered in this study showed the distribution of node degrees in Fig. 65:[3] The x-axis on the horizontal marks the degree of a node (i.e. number of adjacent nodes). The y-axis on the vertical marks the absolute frequency of nodes which have the degree of the corresponding x-value. Thus, the resulting plots display the distribution of node degree frequencies for the considered graphs.

The charts in Fig. 65 illustrate that a considerable number of nodes have only few connections to other nodes. In particular, regions with a degree of 0 are obstacles. Regions with a degree of 1 dead-ends. They are connected to articulation points. As a result this experimental data backs up one of the main hypotheses for the hierarchisation algorithm presented in Sect. 5.4.3: the considered graphs of real floor plans are indeed very sparse despite a remarkable variation in node degrees.

Fig. 66 shows the results of the hierarchisation process on the previous floor plan data: The number of hierarchy levels ranges from one to three. Therefore, Alg. 4 converges quite quickly without creating too many hierarchy levels. The total overhead of the hierarchy is small. Although these first results are promising, the hierarchisation has to be tested on more models to achieve a more complete picture.

[3] Note that the node degree in a multigraph is not defined as the number of incident edges of a node, but as the number of adjacent nodes.

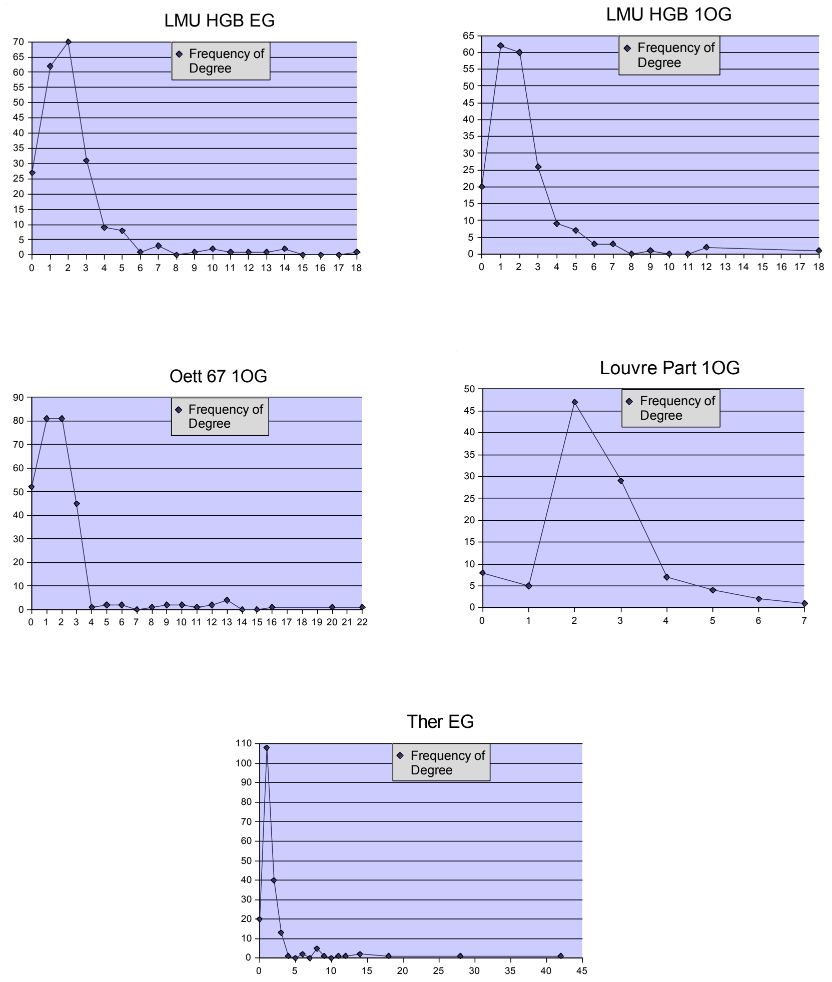


Figure 65: Distribution of Node Degrees in Real Floor Plan Data

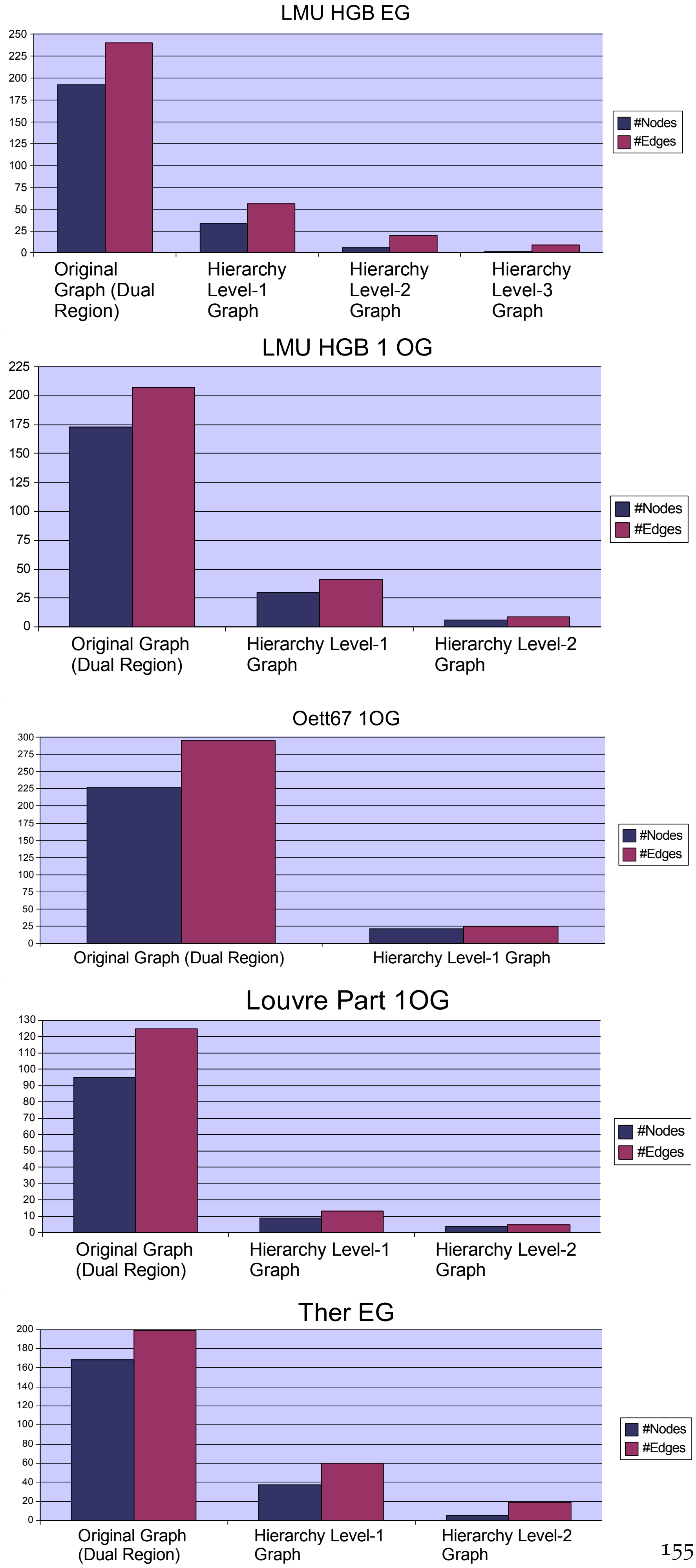


Figure 66: Hierarchisation on Real Floor Plan Data

Part III

CONCLUSION

7 RELATED WORK

This chapter presents work related to the diverse topics covered in this thesis. This concerns, first and foremost, hierarchical graphs and models/systems for indoor navigation. The general topic of wayfinding and models of human wayfinding are also discussed here.

7.1 USE OF HIERARCHIES FOR MODELLING SPATIAL NETWORKS

Classical Applications

Hierarchies of graphs have been introduced and extensively investigated in the fields of Computer Science and Artificial Intelligence mainly because of their computational properties, i.e. to achieve speed-up for path finding in very large graphs. It has been shown that a hierarchical partitioning of large networks into smaller subgraphs can improve the average processing time for network-related queries in practical applications [Sac73, Kno91, HMZM96, SFG97]. In the geospatial domain the particular focus of this research has been, for the most part, on road networks [JHR96, Liu96, HJR00]. These networks offer plenty of commercial applications, such as car navigation systems, traffic surveillance and control, traveller information systems and many more.

Integration into a Geospatial World Model

Note that in all these cases, the hierarchisation is restricted to one type of spatial network. In this thesis however, a novel use case for hierarchical graphs has been explained, namely for the integration of diverse spatial networks. They can all be modelled by (hierarchical) graph structures. To the best knowledge of the author, this use case has been first proposed by Bry et al. [BLOR05]. In a nutshell, the paper describes the following idea: unlike in traditional conceptual modelling, where the co-existence of *multiple* representations generally indicates a design error, multiple representations are quite normal for geographic information [PSZ06]: there can be different maps of the same real world objects at different scales. At a smaller scale, e.g. a building can be represented by a set of floor plans which describe its interior structure. The same building can be represented – at a larger scale – as a node connected by walkways or streets within a city. This can be conveniently modelled in a hierarchical graph structure. Notably, the Open Geospatial Consortium (OGC) has developed various standards for the interoperation of navigation systems and services. At the core, there is the `OpenLS` initiative [Ope], which aims at integrating different navigation systems (and thus leveraging location-based services and route finding). Although the focus is mainly on car navigation and outdoor related scenarios, these ideas can be used as general guidelines for the design of our indoor system. However, a hierarchical structure of graphs was not foreseen in the specification. This is a novel aspect.

Criteria for Hierarchisation

Efficiency was the key issue investigated in classical research on hierarchical graphs. However, little attention has been paid to the problem of (automatically) constructing a hierarchy of graphs and determining the relevant criteria for its construction: in most cases the hierarchy was assumed to be given (e.g. by different districts of a city) or artificially constructed (e.g. by planar graph separators). Whether the hierarchy is plausible for people, i.e. it can be used in communicating spatial knowledge and giving route instructions, has not been examined in these cases. However, alternative criteria for the hierarchisation of a city's street network – which are more akin to human conceptualisations of space – were presented in the Ph.D. thesis of Tomko [Tom07]: for example, the most prominent streets could be determined by their betweenness centrality (see Sect. 7.2.1). Furthermore, prominent landmarks were considered especially because they define areas of influence around them.[1] One can thus say that a region is nearer to a particular landmark than to any other one. The boundaries of these areas of influence around landmarks are the edges of the Voronoi graph generated from the point set of landmarks. This new kind of hierarchisation enhances the communication of route knowledge and alleviates the descriptions of routes through a city.

Meaningful Partitions

Another notable exception can be found in the work of Liu [Liu96]. Here, a partitioning of road networks has been proposed which makes use of the knowledge that roads are divided into different classes/types: In fact, major roads and expressways often partition a road network in a natural way, into different quadrants and triangles. The key idea was that minor roads are often enclosed in such grids formed by major roads. For abstract path planning, only the skeleton of major roads needs to be considered [KTS03]. This phenomenon is also exploited by experienced taxi drivers, e.g. in large metropoles like Mexico City where it is virtually impossible for people to remember all street names. This is a typical case for problem solving by abstraction. Kuipers et al. [KTS03] proposed a similar idea, with the sole difference that boundaries (i.e. linear features such as long roads, rivers or railway tracks) were used for the partitioning. If the endpoints of a route lie on opposite sides of a boundary, it is evident that the boundary has to be crossed eventually on the route. It is therefore an intermediate goal for path finding. A similar heuristic has been proposed in this thesis with articulation points for floor plans.

Foundations of Hierarchical Representations

The theoretical foundations of hierarchical graphs for spatial representation were discussed in Stell [Ste99]. Moreover, Timpf and

[1] Here, landmarks are considered to be point-like shapes.

Frank [TF97] elaborated general principles for carrying out spatial reasoning tasks with hierarchies. Montello [Mon93] classified spatial environments according to their scale – from large ones which are mediated by maps to smaller ones which can be experienced directly, either through locomotion or visual perception. Each of these scales makes up a different conceptual level in a spatial hierarchy. Kuipers et al. [KBG$^+$00] proposed a 'spatial semantic hierarchy' formed by different abstraction levels: it starts from the levels of sensory input, over the causal level (consisting of views and actions), to a topological level with paths, regions and places. The theory is quite general. As such, it can be applied to both robots and humans. Moreover, Plümer et al. [PG96] presented nested maps as hierarchical data structure for spatial models. Nested maps are dual graphs which are hierarchically organised. However, only one edge type is considered in nested maps (i.e. no distinction is drawn between boundaries and openings). Consequently, this model cannot be adopted for indoor environments.

Hierarchy for Buildings

Maio and Rizzi [MR96] introduced in a theoretic work a layered knowledge representation for navigation in buildings, explaining how hierarchical planning can be performed. They also considered the presence of different attributes in their path finding method.

System for a Hospital

Cagigas and Abascal [CA04] have proposed a hierarchical model for indoor environments for an automated guidance system controlling a wheelchair. Their system is called `TetraNauta`. The system was applied to a large hospital building, consisting of several floor levels. For path planning a variant of hierarchical A^* was used and the cost function included additional costs for turns (which require more time for a wheelchair) and elevators for the third dimension. The primary concern in this work was efficient path finding and devising suitable cost functions for the wheelchair.

Hierarchisation of a Floor Plan

Hu and Lee [HL04] also presented a simple hierarchical model for indoor environments. The third dimension was not addressed in their work, though: the model considers only one floor plan. All entries and exits of the floor are considered roots in this hierarchy. Thus the hierarchy is also called 'exit hierarchy'. The other nodes are classified according to their closeness or depth from the roots. The resulting hierarchy is the combination of search trees grown from the exits of the floor. So it is, in general, not a tree-like structure but rather a lattice [LL08], where children have several parents. The duality of locations (regions) and exits is a prominent feature of this model. However, it remains unclear whether the approach really scales, because only toy examples were presented. It needs to be studied further with larger and more complicated examples to give a more realistic account.

7.2 INDOOR NAVIGATION AND WAYFINDING

7.2.1 *General Issues*

Different Data Sets

Pedestrian guidance requires other data sets than car navigation systems, even in outdoor urban areas [CW01, SRK07]. The automated creation of suitable data models is an inherent problem for pedestrian guidance. Elias [Eli07] has proposed a method for this purpose, collecting data from different sources, integrating them and checking their consistency. The issue of constructing a flat graph representation from a floor plan's geometry has been addressed in Whiting et al. [Whi06, WBT07]. A pragmatic approach is presented there. It accounts for various features and also includes openings (called portals) on the boundaries of regions. Among others, problematic situations for graph construction could be identified (e.g. regions which are connected to the exterior). For path finding, a standard approach is adopted. They also considered non-convex regions, for which a Delauney triangulation was performed. But there are also non-standard decompositions considered in other work: Lien and Amato [LA04], for example, perform an approximate convex decomposition, relaxing the strict requirement of convexity. Slightly convex angles are not necessarily decomposed. An interesting approach combining geometric processing and image analysis has been explained in Lefebvre and Hornus [LH03]: they outlined an automatic method for detecting portals of regions and thus decomposing a region into a so called Cell-and-Portal graph.

In the field of qualitative spatial reasoning, various calculi have been devised for describing and deriving topological relations between regions. Among the most prominent are the Region Connection Calculus (RCC-8) by Cohn et al. [CBGG97] and the 9-intersection model by Egenhofer et al. [EF91]. The different abstract relations between regions are depicted in Fig. 67 (using the nomenclature of RCC-8).

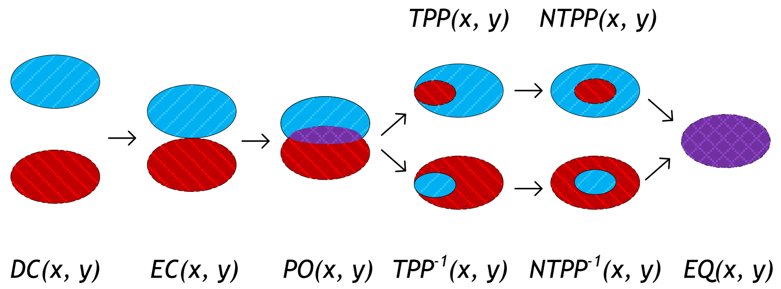


Figure 67: Eight Basic Relations between Regions According to Cohn [CBGG97]

Interestingly, the indoor environments we considered could not be described by the relations of these calculi for two different reasons:

1. there is no multiplicity of relations between regions (for modelling, e.g., two doors between two rooms),[2]

[2] However, this is easy to represent this as multiedges in a dual region graph.

2. there is only one type of connection (i.e. no distinction between opening/portal and boundary).

Ontologies and Wayfinding

For these reasons a more expressive ontology is needed to describe indoor environments in a qualitative manner. In contrast to the general-purpose calculi mentioned above, the work of Bittner [Bit01] features a specific ontology for indoor environments. The ontology provides a thorough background for classifying various boundary types: for example, the boundaries of spatial regions and implicit boundaries within the structure of a spatial region are conceptually distinguished. In the work of Raubal et al. [RW99], a formal wayfinding[3] model is presented based on image schemata and affordances – it describes the sequence of actions which can be performed in an indoor environment, along with their preconditions. An airport environment has been investigated in their studies. Human wayfinding knowledge is internally represented as *cognitive maps* [Gol99]. These maps encode human actions in (and interactions with) an environment, thus make up the essential 'sense' of location. Findings in the research communities of spatial cognition and artificial intelligence [HJ85, Voi03, WM03a, WM03b] suggest that cognitive maps too, are organised in a hierarchy – topological and hierarchical models seem plausible to people.

Wayfinding in Architectural Design

The structure of a building is the result of a well-planned human creation process. Thus it is interesting to see which criteria have been considered in the course of a building's design by its architects. Passini [Pas84, Pas96] was among the first to acknowledge the important role of wayfinding in architectural design. In a recent publication by Brösamle et al. [BH08], interviews have been conducted with architects to analyse existing buildings, discover problems related to wayfinding and propose solutions. It turned out that architects can *anticipate* the *flow* of pedestrians in two different ways: either by reasoning from an allocentric viewpoint about flows and paths or by imagining egocentric walkthroughs. The study tried to reveal the underlying assumptions of architects which are only implicitly stated. Hoelscher et al. have also investigated the correlation between architectural design and human wayfinding behaviour [HMV$^+$04, HMV$^+$06]. They carried out various experiments. Amongst others, wayfinding strategies could be identified for multi-level buildings. These findings back up diverse hierarchisations of floor plans, providing experimental support that wayfinding behaviour is influenced by architectural design. Complementary to this, further experiments have been conducted by Wiener et al. [WFR$^+$07] with a set of virtual indoor scenes. They investigated the navigation behaviour of people and the relation to parts of regions which are formed by coherent visibility (called *'isovists'*).

[3] Notably, the notion of wayfinding comprises more than just path finding – human wayfinding is the complete process of locomotion, orientation, spatial reasoning and problem solving. The term has been coined by Lynch in *'The Image of the City'* [Lyn60] and is widely used ever since. In accordance with the viewpoint of Passini, wayfinding can be considered as spatial problem solving and decision making [Pas84, Pas96].

Such findings play a role for the decomposition of non-convex regions. The geometric layout of architecture was also studied by Peponis et al. [PZC90, Pep97, PWR+97], who proposed different principles for decomposing spaces.

Flow Simulation

In order to gain information on the anticipated flow of pedestrians through a building, one can resort to techniques from the field of graph theory, e.g. simulating flows starting from entries and diffusing into the interior with a certain probability. Turner and Penn [TP02], for example, propose an agent-based system which makes use of Markov chains in a stochastic process, assigning probabilities to transitions. The simulation has been applied to the famous Tate Gallery in London, see Fig. 68.

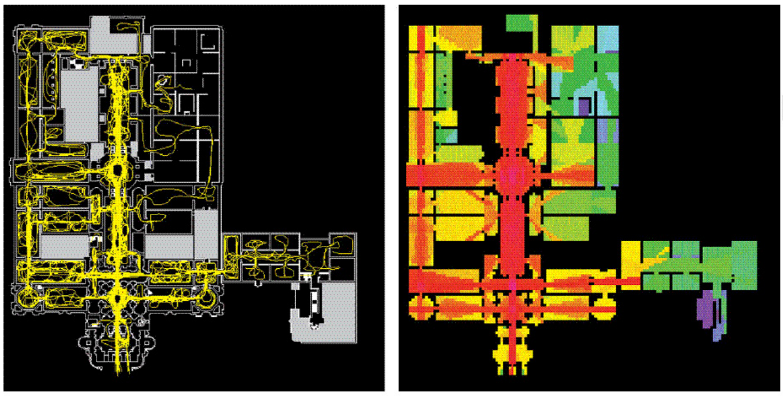


Figure 68: Traces of Pedestrian Movement in the Tate Gallery [Dur07]

Network Analysis

Another possible approach in this field is to apply techniques of network analysis. These are, for example, employed in large social networks. There are a series of various centrality measures which can be used for this purpose [Sab66, Fre77]:

- *degree centrality* is simply the number of connections from a given node. The problem with *degree centrality* is that it accounts only for local prominence. For example, a long corridor with many offices at one end of the building may be considered as central merely due its large amount of connections.
- *closeness centrality* measures the mean geodesic distance (i.e., shortest path length) of a node to *all* other nodes in the network. Intuitively, the nodes which are the least far away from all others are considered central nodes.
- *betweenness centrality* of a node N is the relative frequency of its occurrences within all shortest paths through the network (for origins and destinations not including N). Note that there are $(n-1) \times (n-2)$ paths possible since paths are generally not symmetrical, but direction may be of importance. If a node occurs in many shortest paths, it is likely that it also appears as an intermediate goal for a particular path. Therefore, this centrality value qualifies as a good indicator for importance in terms of wayfinding.

Modalities

The work of Hendricks et al. [HEH03] is of particular theoretical interest. They propose to structure the wayfinder's environment in a qualitative and cognitively plausible manner: there are compulsions (must-goals) and barriers (avoid-goals) as primitives. An interesting connection is created between the spatial structure and the implied *modalities* for the wayfinder. Modalities can, for example, be expressed in verbs such as 'can', 'may' or 'must' and their respective negations. The separation into public and private spaces (cf. Sect.2.2.5) can also be represented by modalities ('may' or 'may not'). In our hierarchisation, articulation points in a graph (see Sect. 5.4.3.2) can be seen as must-goals/compulsions if they are on a path between origin and destination. Obstacles are consequently avoid-goals, but also all regions inaccessible due to access restrictions or other constraints (see Sect. 4.1.3: physical: cannot, social/legal: may not).

7.2.2 *Systems and Approaches*

In addition to the `TetraNauta` system described in Sect. 7.1, there are a couple of other research prototypes and interesting approaches for indoor navigation systems specifically designed for pedestrian guidance:

System of Merminod et al.

Merminod [GM03] has worked out the main components in the architecture of an indoor navigation system for pedestrians and presented a prototypical system. He outlined an approach based on CAD databases for indoor maps from which a topological graph structure (equivalent to flat dual region graphs) was extracted. Emphasis was put on the integration with indoor positioning systems, implementing different map matching[4] algorithms. A combination of different sensors was used for positioning (including a gyroscope for determining the user's orientation, a digital magnetic compass and a GPS receiver). These sensors have been integrated into a small device. Moreover, constraints such as access restrictions have also been considered. In the implementation, these are represented as attributes which are attached to nodes and edges of the graph.

The Yamamoto Toolkit

The `Yamamoto` [Sta, SH06] system has been developed as a map modelling toolkit for indoor environments. With this tool one can create vector-based representations of the floor plans of a building, overlay them with a scanned image of the building and georeference the map. It also offers an appealing graphical user interface with a 3D rendering engine. One specific feature of the system is the so called avatar view, which is an egocentric view for exploring the environment in 3D (shown in Fig 69). Maps of floor plans are stored in an XML-based format. It was therefore convenient to use the Yamamoto editor in this thesis for creating maps of indoor spaces.

Route instructions are also provided by Yamamoto in graphical form [MS07]: Different arrows in the avatar view indicate the direction to continue.

[4] assigning the measured position to a digital map

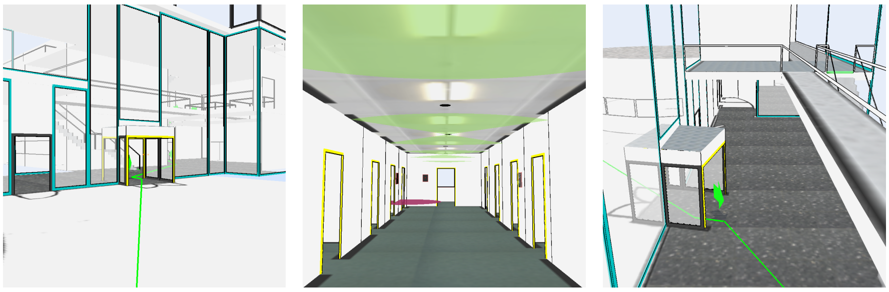


Figure 69: Virtual Walk Through a Building with Yamamoto [Sta, MS07]

OntoNav System

Moreover, Tsetsos et al. [TAK$^+$05, TAKH06] proposed a semantic model for indoor environments based on ontologies. The system is called `OntoNav`. It is 'human-centered' in the sense that personal preferences are taken into account for path selection by rules. More precisely, boolean constraints can be specified by means of SWRL rules. Their idea of using ontologies for specifying context information has been adopted in this thesis. However, one significant improvement has been made compared to the original system of Tsetsos et al.: context information is not evaluated *after* a path finding algorithm has been applied, but instead *before* or *during* path finding (cf. Sect. 4.1.3). Apart from this, the underlying geometry of floor plans is an aspect neglected in the OntoNav system.

Wiki for Context Information

Another possibility for dealing with context information is presented in Whiting et al. [Whi06, WBT07]: the main idea is to let users specify and share context information in a Wiki. This should make it easier for people or groups to define their own knowledge relevant for path finding.

Route Descriptions

Simple means for describing paths in a way comprehensible to people is another important matter for indoor navigation. For this purpose, Dale et al. [DGP03, DGP05] have developed the `Coral` system which resorts to techniques for Natural Language Generation (NLG). Their system is however tailored for the simpler domain of road networks. Improvements of the Coral system, together with an adaptation to indoor environments have been proposed by Mizzi and Rosner [Miz04, RM04]. The quality of these route descriptions is due to a preceding analysis of large corpora of textual route descriptions given by people. Like in the Yamamoto system, a simple graphical editor is provided for creating floor plans. These plans are, analogously, stored in an XML representation. Although the generation of route descriptions is elaborate from a linguistic point of view, the considered geometry of floor plans is rather simple: only rectangular shapes are allowed.

Following a similar direction of research, a route guidance grammar has been presented by Prusi et al. [PKH$^+$05]. Another linguistic approach for obtaining good route descriptions was presented by Look et al. [LKLS05]. However, a major drawback is that this approach is not yet fully automated. The opposite direction was considered by MacMahon et al. [MSK06]: from a given textual route description, the task was to interpret these instructions and let an agent follow them automatically in a virtual indoor environment. The idea to combine Generalised Voronoi Graphs with orientation calculi for route descriptions was put forth by Krieg-Brückner et al. [KBS06]. Richter and Klippel [RK04, RK06, Ric07] investigated different means to structure route descriptions around landmarks and compress them into a more concise representation (i.e. so called methods of *spatial chunking*).

Region Detection

Complementary to the work presented in this thesis, a substantial body of research on region detection [RsMMSB05, sMMTJ^{+}07, PSN08] can be found both in the image processing and robotics communities. The goal is to build a geometric model of a building and subsequently a graph model from non-CAD data, mostly from sensor data obtained by robots from laser scans. In the course of this process, image data is also analysed and transformed into a vector-based format which is more common for CAD systems. This aspect has been left out in the discussions and ignored so far. However, the interpretation of architectural drawings [LSM98] is somewhat more sophisticated because symbols have to be distinguished from actual geometry. For this reason we have left out the interpretation of drawings and non-vector data on purpose. Instead we concentrated on geometric models more akin to those obtained by CAD systems (i.e. vector-based). Nevertheless, considering the detection of regions by different means clearly broadens the spectrum of possible sources of data to be fed into our system. Any drawing of a floor plan could then be interpreted in an appropriate manner, resulting in a very powerful system.

8 SUMMARY AND FUTURE WORK

In this chapter, the results presented in this thesis are summarised and possible directions for future research building on this work are suggested.

8.1 RESULTS

The previous chapters illustrated the construction of hierarchic graph structures and their use for indoor navigation.

Summarising, the main results of this work are:

- In Chapter 4, hierarchical graphs were introduced as a generic spatial data infrastructure. In particular, this architecture allows for integrating different spatial networks which originate from different sources. A small, but useful set of operations has been proposed for integrating networks. The result is a highly extensible system in which different graphs/networks can be added for a more comprehensive navigation across different networks. A prototypical implementation of this architecture has been carried out as a proof-of-concept.

- In Chapter 5, a systematic method has been presented to automatically obtain a hierarchical graph model from a set of floor plans. Two novel aspects were discussed in this respect: Firstly, the unique characteristics of indoor environments can be incorporated into the hierarchisation process to enhance classical graph search techniques on flat graphs. The resulting hierarchy can be used as an informed heuristic to reduce search space. At the same time, elements in this hierarchy represent meaningful subgraphs, such as sections or wings of a building. The algorithm for constructing this hierarchy has been implemented and its validity has been confirmed by a study of real floor plan data. Secondly, the hierarchy can be exploited for giving directions in irregular and complex-shaped regions. We have outlined a method for deriving qualitative spatial relations from a region's geometry which can be used in this process.

Together, the proposed methods and algorithms describe the basic principles of an indoor navigation system. Although most of the basic methods for hierarchisation were implemented and could be tested on several building models of the university with some promising results (see Chapter 6), this is just a preliminary evaluation.

A comprehensive study in a realistic application context would be required to evaluate their performance in practice and propose further enhancements and options for fine-tuning. Unfortunately this was not possible within the limited time of this research. The presented methods and algorithms can nevertheless be used as guidelines for future research.

8.2 CONCLUSION

Recapitulating, three main goals were set out in the introduction (see Sect. 1.2). In the course of this thesis we have gained the following insights on them:

Integration

The first goal of integration has been thoroughly explored in Chapter 4 with the proposal of a system architecture and a proof-of-concept implementation. We have argued that hierarchical graphs are eminently suitable for the purpose of integration.

Automation

Concerning the second goal of automation, Chapter 5 detailed a systematic method for constructing a hierarchical graph from a set of floor plans and showed problematic situations from a geometrical or conceptual modelling perspective, including the third dimension. Different floor plans of real buildings were analysed for this purpose. Another important outcome was that automation can be only done to a certain degree. We have discussed problematic cases and thus the inherent limits of this automation process.

Suitability for Humans

The third goal, the conception of a human-friendly system, has had a strong influence on various aspects of the system. Suitability for humans is a key requirement for the design of a pedestrian indoor navigation system. Thus it is reflected our system design: first, the specification of context information and its integration into the path finding process (see Sect. 4.1.3) can be seen as an important contribution for improving navigation systems by making them more flexible and adaptable. The idea is to better cope with individual requirements of users. Moreover, two hierarchisation methods have been proposed in Sect. 5.4 to meet the special requirements of *pedestrian* indoor navigation. They deal with the shortcomings of the first version of our model which have come up by analysing a couple of examples from real floor plans. As explained in Sect. 5.5, we can use this hierarchy not only to enhance path finding, but – more importantly – to generate meaningful route descriptions from it. What is still missing is an evaluation of route descriptions by human users to reveal the strengths and weaknesses of our approach.

8.3 DIRECTIONS FOR FUTURE RESEARCH

The work on a hierarchical graph system for indoor navigation is far from finished. There are a number of possibilities for extending the overall system and enhancing its individual components. Only the main directions and ideas are presented here:

Pattern Recognition

The geometric shapes of regions in floor plans could be evaluated with various methods of pattern recognition. This way one could automatically classify regions – according to their shape and dimensions – into different classes. This method could be used not only to distinguish between rooms, corridors and halls, but also in combination with a geometric decomposition algorithm for non-convex regions. In the latter case one could analyse different types of corners, junctions and intersections. They are all regions resulting from a geometric decomposition.

Further Use of Ontologies

Moreover, one could use this classification of spaces and automatically create concepts in an underlying ontology for the elements of a building. These concepts can also be used as landmarks for route descriptions. One could also specify in an ontology of building elements which nestings of different region types are allowed (e.g. rooms as parts of a floor or a section and not vice versa). This would also ensure the *semantic* consistency of a hierarchical graph system.

Ontology-based Route Descriptions

Generating route descriptions is another challenging topic. The methods and ideas presented in this thesis are just a starting point. They can be further studied and extended. The ultimate goal is to obtain a language or ontology for formal route descriptions, instead of a simple XML representation. This language should encode different actions of human wayfinders in a qualitative manner, so that these can be easily transformed into verbal descriptions by using techniques of natural language generation.

Evaluation of Route Descriptions

User studies would be needed to assess the general quality of route descriptions for this purpose. There are generally two possibilities: one could conduct experiments for testing the descriptions in a real environment. On the one hand this is more difficult to manage, but on the other hand this situation is the most realistic. Another option is to conduct the experiments in a virtual environment. The Yamamoto system by Stahl et al. [Sta, SH06], which has been used in this thesis for the creation of floor plan data, can also be used in this respect: it offers an avatar view for a virtual walk through in an indoor model (see Fig. 69 in Sect. 7.2.2). Thus one could also evaluate instruction-following in this virtual environment.

Hierarchical Optimisation

Different algorithms for hierarchical optimisation could be studied on the floor plans of a building and compared in a number of respects, such as time consumption, number of expanded nodes during search, robustness and ability to deal with dynamic changes in the graph. The results and insights gained from such a thorough analysis could be used to improve the hierarchisation process further in this respect. But one could also exploit a hierarchical graph structure for a variety of more complex problems. For example, the hierarchic structure could be used as a heuristic for travelling salesmen problems inside a building. This could be useful for planning a visit in a museum. With limited time available, visitors can hardly manage to see everything. Rather, a tour could be suggested which takes into account the individual preferences of users and tries to maximise the coverage of must-see goals. Moreover, different evaluation strategies for constraints and context information in a hierarchical graph structure can be further investigated.

A

ONTOLOGY FOR CONTEXT INFORMATION

The specification of the ontology for context information presented in Sect. 4.1.3 is shown below. It was created by a student under the supervision of the author. For the complete implementation details, interested readers can look up the related project and diploma thesis [Ide06, Ide07].

```
Namespace: =<http://www.owl-ontologies.com/Ontology1153678363.owl#>
Namespace: rdfs = <http://www.w3.org/2000/01/rdf-schema#>
Namespace: owl11 = <http://www.w3.org/2006/12/owl11#>
Namespace: owl11xml = <http://www.w3.org/2006/12/owl11-xml#>
Namespace: Ontology1153678363 = <http://www.owl-ontologies.com/Ontology1153678363.owl#>
Namespace: owl = <http://www.w3.org/2002/07/owl#>
Namespace: xsd = <http://www.w3.org/2001/XMLSchema#>
Namespace: rdf = <http://www.w3.org/1999/02/22-rdf-syntax-ns#>


Ontology: <http://www.owl-ontologies.com/Ontology1153678363.owl>

ObjectProperty: allowed_TransportationDevices

    Annotations:
        rdfs:comment ""
    Domain:
        DriverLicence
    Range:
        SelfPoweredTransportationDevice
    Inverses:
        driving_licence_needed


ObjectProperty: gender

    Characteristics:
        Functional
    Domain:
        Person_Basic_Data
    Range:
        Gender


ObjectProperty: owns_private_transp_vehicles

    Domain:
        Transportee_Possibilities
    Range:
        Private_Transportation


ObjectProperty: drives_TransportationDevice
```

```
    Characteristics:
        Functional
    Domain:
        Driver
    Range:
        TransportationDevice


ObjectProperty: hasPossibilities

    Domain:
        Transportee
    Range:
        Transportee_Possibilities


ObjectProperty: position_target

    Annotations:
        rdfs:comment "geographical position of the Transportee's destination"
    Domain:
        Transportee


ObjectProperty: driving_licence_needed

    Characteristics:
        Functional
    Domain:
        SelfPoweredTransportationDevice
    Range:
        DriverLicence
    Inverses:
        allowed_TransportationDevices


ObjectProperty: hasRestrictions

    Annotations:
        rdfs:comment "restrictions limiting the Transportee's available/usable means of transportation"
    Domain:
        Transportee
    Range:
        Transportee_Restrictions


ObjectProperty: driver_licence

    Domain:
        Transportee_Possibilities
    Range:
        DriverLicence


ObjectProperty: user_transportation_preferences

    Domain:
        Transportee
    Range:
        Transportation_Preferences

```

```
ObjectProperty: person_basic_data

    Characteristics:
        Functional
    Domain:
        Person
    Range:
        Person_Basic_Data


ObjectProperty: relatedMeansOfTransportation

    Annotations:
        rdfs:comment "means of transportation a transportee
             has access to (does not mean that he can also
            use it -> restrictions). could be used e.g. for
            loading the set of public transportation devices
             into a user profile depending on the current
            city the user is located. or for specifying the
            exact model of the users private car."
    Domain:
        Transportee
    Range:
        Mean_of_Transportation


ObjectProperty: available_TransportationDevices

    Annotations:
        rdfs:comment "TransportationDevices a Transportee
            can use; devices could be limited depending on
            the restriction class"
    Domain:
        Transportee_Restrictions
    Range:
        Mean_of_Transportation


ObjectProperty: can_transport_other_TransportationDevices

    Annotations:
        rdfs:comment "availability to transport
            TransportationDevices, e.g. a train can
            transport a bike"
    Domain:
        TransportationDevice
    Range:
        TransportationDevice


ObjectProperty: uses_transport_network

    Annotations:
        rdfs:comment "which infrastructure is needed for
            moving, e.g. road for cars"
    Domain:
        Mean_of_Transportation


ObjectProperty: homeaddress
    Domain:
        Person_Basic_Data
```

```

ObjectProperty: position_current

    Annotations:
        rdfs:comment "geographical position of the
            Transportee"
    Domain:
        Transportee


ObjectProperty: luggage_properties

    Annotations:
        rdfs:comment "this property contains a luggage
            individual for each luggage item"
    Domain:
        Luggage_Restriction
    Range:
        Luggage


ObjectProperty: monthly_tickets_public_transport

    Domain:
        Transportee_Possibilities
    Range:
        Public_Transportation


DataProperty: cost_per_kilometer

    Characteristics:
        Functional
    Domain:
        Kinds_of_Transportation
    Range:
        float


DataProperty: cost_per_kilometer_additionalBasicPrice

    Characteristics:
        Functional
    Domain:
        Taxi
    Range:
        float


DataProperty: taxi_phone_number

    Characteristics:
        Functional
    Domain:
        Taxi
    Range:
        int


DataProperty: dimensions_height

    Characteristics:
        Functional
    Domain:
        Luggage
        or TransportationDevice
    Range:
        int


DataProperty: mobilephone_number

    Characteristics:
        Functional
    Domain:
        Person_Basic_Data
    Range:
        int


DataProperty: dimensions_width

    Characteristics:
        Functional
    Domain:
        Luggage
        or TransportationDevice
    Range:
        int


DataProperty: weight_kg

    Characteristics:
        Functional
    Domain:
        Luggage
        or TransportationDevice
    Range:
        int


DataProperty: available_till_time

    Characteristics:
        Functional
    Domain:
        Public_Transportation
    Range:
        string


DataProperty: model_name

    Characteristics:
        Functional
    Domain:
        Automobile
    Range:
        string


DataProperty: maximum_luggage_height

    Characteristics:
        Functional
    Domain:
        Mean_of_Transportation
    Range:
        int


DataProperty: free_stopping_point

    Characteristics:
        Functional
    Domain:
        Kinds_of_Transportation
    Range:
        boolean


DataProperty: maximum_luggage_weight

    Characteristics:
        Functional
    Domain:
        Mean_of_Transportation
    Range:
        int


DataProperty: available_from_time

    Characteristics:
        Functional
    Domain:
        Public_Transportation
    Range:
        string


DataProperty: vehicle_id

    Characteristics:
        Functional
    Domain:
        SelfPoweredTransportationDevice
    Range:
        string


DataProperty: transportation_preference

    Characteristics:
        Functional
    Domain:
        Transportation_Preferences
    Range:
        float


DataProperty: line_number

    Characteristics:
        Functional
    Domain:
        RailVehicles
        or Bus
    Range:
        string


DataProperty: maximum_speed_kmh

    Characteristics:
        Functional
    Domain:
        Mean_of_Transportation
    Range:
        int


DataProperty: age

    Characteristics:
        Functional
    Domain:
        Person_Basic_Data
    Range:
        int


DataProperty: cost_per_hour

    Characteristics:
        Functional
    Domain:
        Kinds_of_Transportation
    Range:
        int


DataProperty: free_starting_point

    Characteristics:
        Functional
    Domain:
        Kinds_of_Transportation
    Range:
        boolean


DataProperty: capacity

    Characteristics:
        Functional
    Domain:
        Mean_of_Transportation
    Range:
        int


DataProperty: maximum_luggage_length

    Characteristics:
        Functional
    Domain:
        Mean_of_Transportation
    Range:
        int
```

```

DataProperty: name_first

    Characteristics:
        Functional
    Domain:
        Person_Basic_Data
    Range:
        string


DataProperty: maximum_luggage_width

    Characteristics:
        Functional
    Domain:
        Mean_of_Transportation
    Range:
        int


DataProperty: dimensions_length

    Characteristics:
        Functional
    Domain:
        Luggage
        or TransportationDevice
    Range:
        int


DataProperty: usable_for_disabled_persons

    Characteristics:
        Functional
    Domain:
        TransportationDevice
    Range:
        boolean


DataProperty: name_last

    Characteristics:
        Functional
    Domain:
        Person_Basic_Data
    Range:
        string


DataProperty: transportation_type

    Characteristics:
        Functional
    Domain:
        Transportation_Preferences
    Range:
        string

```

```
DataProperty: email_address

    Characteristics:
        Functional
    Domain:
        Person_Basic_Data
    Range:
        string


DataProperty: childseat

    Characteristics:
        Functional
    Domain:
        Bicycle
    Range:
        boolean


Class: BlindPerson

    Annotations:
        rdfs:comment "blind person"
    SubClassOf:
        Disabilities_Restriction,
        available_TransportationDevices only (Tram_new
                                              or Bus
                                              or
                                                  Pedestrian

                                              or S_Train
                                              or Tram_old
                                              or U_Train
                                              or Taxi)


Class: Tram_new

    Annotations:
        rdfs:comment "Tram with no stairs an entry,
            disability friendly;"
    SubClassOf:
        Tram
    DisjointWith:
        Tram_old


Class: Young_children

    Annotations:
        rdfs:comment "babies or young children who can not
            travel alone"
    SubClassOf:
        Additional_Persons_Restriction,
        available_TransportationDevices only (Tram_new
                                              or Bus
                                              or Van
                                              or Truck
                                              or
                                                  PrivateCar
                                              or
                                                  Pedestrian

                                              or S_Train
                                              or Tram_old
                                              or U_Train
                                              or Taxi)


Class: UserPoweredTransportationDevice

    Annotations:
        rdfs:comment "TransportationDevice without own
            power source, device must be powered by the user
            "
    SubClassOf:
        TransportationDevice


Class: Wheelchair

    Annotations:
        rdfs:comment "a Transportee who can not use his
            legs and is tied to an wheelchair; can use his
            car or van if it is special prepared;"
    SubClassOf:
        Disabilities_Restriction,
        available_TransportationDevices only (Tram_new
                                              or Bus
                                              or Van
                                              or
                                                  PrivateCar

                                              or
                                                  Pedestrian

                                              or S_Train
                                              or U_Train
                                              or Taxi)


Class: Luggage

    Annotations:
        rdfs:comment "things a transportee wants to move
            from one point to an other (except of additional
             escorting persons or animals which are not
            transported in a cage)"


Class: Motorcycle

    Annotations:
        rdfs:comment "two-wheeled road vehicle designed for
             carrying one or two passengers"
    SubClassOf:
        Private_Transportation
    DisjointWith:
        Truck,
        Van,
        Car,
        Pedestrian,
        Bicycle
```

```


Class: DriverLicence

    Annotations:
        rdfs:comment "Document needed for driving a vehicle
            "


Class: Tram_old

    Annotations:
        rdfs:comment "Tram with stairs an entry, disability
             unfriendly;"
    SubClassOf:
        Tram
    DisjointWith:
        Tram_new


Class: EUDriver_C

    Annotations:
        rdfs:comment "european driver licence for trucks
            (>3,5t)"
    SubClassOf:
        DriverLicencePrivateTransport,
        allowed_TransportationDevices only Truck


Class: Gender

    Annotations:
        rdfs:comment "male or female"
    EquivalentTo:
        {Female , Male}


Class: SelfPoweredTransportationDevice

    Annotations:
        rdfs:comment "TransportationDevice which runs with
            own power source; alternative name: vehicle"
    SubClassOf:
        TransportationDevice


Class: Taxi

    Annotations:
        rdfs:comment "Car which always includes a driver
            who always belongs to the car; one to four
            additional persons can use this vehicle;"
    SubClassOf:
        Car


Class: Mean_of_Transportation

    Annotations:
        rdfs:comment "technical view on moving an object (
            person and/or luggage) from one point to another
            "
```

```

Class: Person_Basic_Data

    Annotations:
        rdfs:comment "basic informations about a person"


Class: DriverLicence_S_Train

    Annotations:
        rdfs:comment "licence for S_Train driver"
    SubClassOf:
        DriverLicencePublicTransport,
        allowed_TransportationDevices only S_Train


Class: Kinds_of_Transportation

    Annotations:
        rdfs:comment "usage of the Mean_of_Transportations"


Class: Bicycle

    Annotations:
        rdfs:comment "two-wheeled
            UserPoweredTransportationDevice for carrying one
             or two persons"
    SubClassOf:
        UserPoweredTransportationDevice
    DisjointWith:
        Truck,
        Van,
        Car,
        Pedestrian,
        Motorcycle


Class: RailVehicles

    Annotations:
        rdfs:comment "vehicles moving on rails"
    SubClassOf:
        SelfPoweredTransportationDevice


Class: Escort_for_disabled_persons

    Annotations:
        rdfs:comment "a Person who guides a Transportee or
            supports the Transportee in transportation"
    SubClassOf:
        Additional_Persons_Restriction,
        available_TransportationDevices only (Tram_new
                                              or Bus
                                              or Van
                                              or Truck
                                              or
                                                  PrivateCar
                                              or
                                                  Pedestrian
                                              or S_Train
                                              or Tram_old
                                              or U_Train
                                              or Taxi)


Class: Truck

    Annotations:
        rdfs:comment "road vehicle designed for carrying
            one to three passengers; mainly designed for
            carrying large luggage;"
    SubClassOf:
        RoadVehicles
    DisjointWith:
        Van,
        Car,
        Pedestrian,
        Bicycle,
        Motorcycle


Class: Transportation_Preferences

    Annotations:
        rdfs:comment "rating of a specific
            Mean_of_Transportation if it is preferred or
            disliked;"


Class: Transportee_Restrictions

    Annotations:
        rdfs:comment "properties that restrict using means
            of transportation"


Class: S_Train

    Annotations:
        rdfs:comment "rail vehicle for city and suburban
            connections; own railway system, moves usually
            inner city underground and outer city overground
            "
    SubClassOf:
        RailVehicles
    DisjointWith:
        Tram,
        Bus,
        U_Train


Class: U_Train

    Annotations:
        rdfs:comment "rail vehicle for city connections;
            own railway system, moves usually inner city
            underground;"
    SubClassOf:
        RailVehicles
    DisjointWith:
        Tram,
        Bus,
        S_Train


Class: DriverLicencePublicTransport

    Annotations:
        rdfs:comment "licence for vehicles with maximum capacity of 10 persons and more; vehicles belong to the class of Public_Transportation"
    SubClassOf:
        DriverLicence


Class: Tram

    Annotations:
        rdfs:comment "rail vehicle for inner city connections; railway system always overground and usually integrated in the urban street network; has to follow the official street traffic rules;"
    SubClassOf:
        RailVehicles
    DisjointWith:
        Bus,
        S_Train,
        U_Train


Class: EUDriver_B

    Annotations:
        rdfs:comment "european driver licence for automobiles (maximum 3,5t)"
    SubClassOf:
        DriverLicencePrivateTransport,
        allowed_TransportationDevices only Automobile


Class: Transportee

    Annotations:
        rdfs:comment "Persons who moves from one point to another"
    SubClassOf:
        Person


Class: DriverLicence_Tram

    Annotations:
        rdfs:comment "licence for tram driver (Tram_old and Tram_new)"
    SubClassOf:
        DriverLicencePublicTransport,
        allowed_TransportationDevices only Tram


Class: Small_Luggage_Restriction
```

```
    Annotations:
        rdfs:comment "bags, small cases, small packages,
            <10kg"
    SubClassOf:
        Luggage_Restriction


Class: Person

    Annotations:
        rdfs:comment "individual human being"


Class: DriverLicence_U_Train

    Annotations:
        rdfs:comment "licence for U_Train driver"
    SubClassOf:
        DriverLicencePublicTransport,
        allowed_TransportationDevices only U_Train


Class: Animal_Restriction

    Annotations:
        rdfs:comment "animals which escorts a Transportee,
            usually tied to the Transportee with a chain;
            animals in cages belong to the class of luggage;
            "
    SubClassOf:
        Transportee_Restrictions,
        available_TransportationDevices only (Tram_new
                                              or Bus
                                              or Van
                                              or Truck
                                              or
                                                 PrivateCar
                                              or
                                                 Pedestrian
                                              or S_Train
                                              or Bicycle
                                              or Tram_old
                                              or U_Train
                                              or Taxi)


Class: TransportationDevice

    Annotations:
        rdfs:comment "artifact designed to move an object
            from one location to another"
    SubClassOf:
        Mean_of_Transportation


Class: Pedestrian

    Annotations:
        rdfs:comment "Persons walking from one point to
            another"
    SubClassOf:
        Private_Transportation
    DisjointWith:
        Van,
        Truck,
        Car,
        Bicycle,
        Motorcycle


Class: Private_Transportation

    Annotations:
        rdfs:comment "personal determined transportation;
            no tickets needed; if a vehicle is used, it is
            usually owned or rent by the transportee or one
            of his companies; maximum capacity 8 persons + 1
             driver"
    SubClassOf:
        Kinds_of_Transportation


Class: Emergency

    SubClassOf:
        Kinds_of_Transportation


Class: Transportee_Possibilities

    Annotations:
        rdfs:comment "additional means of transportation a
            Transportee can use because of licenses, owning
            vehicles or tickets for Public_Transportation;"


Class: Driver

    Annotations:
        rdfs:comment "Transportee who drives the
            TransportationDevice which transports him"
    SubClassOf:
        Transportee


Class: Bus

    Annotations:
        rdfs:comment "road vehicle designed for carrying
            about up to 80 passengers; four or six wheels;"
    SubClassOf:
        Public_Transportation
    DisjointWith:
        Tram,
        S_Train,
        U_Train


Class: Car

    Annotations:
        rdfs:comment "automobile for one to five persons,
            maximum luggage usually limited;"
    SubClassOf:
        Automobile
    DisjointWith:
        Truck,
        Van,
        Pedestrian,
        Bicycle,
        Motorcycle


Class: Additional_Persons_Restriction

    Annotations:
        rdfs:comment "Persons that are bind to a
            Transportee; Transportee and additional Person
            can not be divided while transportation;"
    SubClassOf:
        Transportee_Restrictions


Class: EUDriver_A

    Annotations:
        rdfs:comment "european driver licence for
            motorcycles"
    SubClassOf:
        DriverLicencePrivateTransport,
        allowed_TransportationDevices only Motorcycle


Class: Extralarge_Luggage_Restriction

    Annotations:
        rdfs:comment "extra large packages, furniture, etc.
            "
    SubClassOf:
        Luggage_Restriction,
        available_TransportationDevices only (Truck
                                              or Van)


Class: PrivateCar

    Annotations:
        rdfs:comment "Car which is driven by the
            Transportee; maximum four additional persons can
             join;"
    SubClassOf:
        Car


Class: Luggage_Restriction

    Annotations:
        rdfs:comment "Transportee wants to move luggage
            from one point to another; restricted use of
            Mean_of_Transportation because of the maximum
            luggage size which can be transported;"
    SubClassOf:
        Transportee_Restrictions


Class: Large_Luggage_Restriction

```

```
    Annotations:
        rdfs:comment "more than 3 suitcases, packages over
            20kg, medium size furniture"
    SubClassOf:
        Luggage_Restriction,
        available_TransportationDevices only (Van
                                             or Truck
                                             or
                                                 PrivateCar

                                             or Taxi)



Class: Automobile

    Annotations:
        rdfs:comment "four-wheeled road vehicle designed
            for carrying about one to eight passengers"
    SubClassOf:
        RoadVehicles


Class: Van

    Annotations:
        rdfs:comment "automobile for two to five eight,
            maximum luggage much higher in comparison to a
            car;"
    SubClassOf:
        Automobile
    DisjointWith:
        Truck,
        Car,
        Pedestrian,
        Bicycle,
        Motorcycle


Class: Public_Transportation

    Annotations:
        rdfs:comment "personal determination of
            transportation is limited to a fixed starting/
            endpoint and time; capacity is above 10 persons
            and usually more than 10 transportees use the
            same object at the same time; transportee needs
            ticket for using it;"
    SubClassOf:
        Kinds_of_Transportation


Class: EUDriver_D

    Annotations:
        rdfs:comment "european driver licence for bus
            driver"
    SubClassOf:
        DriverLicencePublicTransport,
        allowed_TransportationDevices only Bus


Class: Medium_Luggage_Restriction
```

```

    Annotations:
        rdfs:comment "suitcases (up to 2), packages (<20kg)
            "
    SubClassOf:
        Luggage_Restriction,
        available_TransportationDevices only (Tram_new
                                              or Bus
                                              or Van
                                              or Truck
                                              or
                                                  PrivateCar

                                              or
                                                  Pedestrian

                                              or S_Train
                                              or Tram_old
                                              or U_Train
                                              or Taxi)



Class: Disabilities_Restriction

    Annotations:
        rdfs:comment "Transportee who is restricted in
            using Means_of_Transportation because of
            personal disabilities;"
    SubClassOf:
        Transportee_Restrictions


Class: RoadVehicles

    Annotations:
        rdfs:comment "vehicles moving on streets"
    SubClassOf:
        SelfPoweredTransportationDevice


Class: DriverLicencePrivateTransport

    Annotations:
        rdfs:comment "licence for vehicles with maximum
            capacity of 8 persons + driver; vehicles belong
            to the class of Private_Transportation"
    SubClassOf:
        DriverLicence


Individual: User_2

    Types:
        Transportee
    Facts:
        relatedMeansOfTransportation  Truck_1,
        relatedMeansOfTransportation  Tram_old_1,
        relatedMeansOfTransportation  S_Train_1,
        relatedMeansOfTransportation  PrivateCar_1,
        relatedMeansOfTransportation  Bus_1,
        relatedMeansOfTransportation  U_Train_1,
        relatedMeansOfTransportation  Pedestrian_1,
        relatedMeansOfTransportation  Bicycle_1,
        relatedMeansOfTransportation  Tram_new_1,
        relatedMeansOfTransportation  Van_1,
        relatedMeansOfTransportation  Motorcycle_1,
        relatedMeansOfTransportation  Taxi_1,
        user_transportation_preferences
            Transportation_Preferences_User_2_u_train


Individual: User_1

    Types:
        Transportee
    Facts:
        relatedMeansOfTransportation  Truck_1,
        relatedMeansOfTransportation  Tram_old_1,
        relatedMeansOfTransportation  S_Train_1,
        relatedMeansOfTransportation  PrivateCar_1,
        relatedMeansOfTransportation  Bus_1,
        relatedMeansOfTransportation  U_Train_1,
        relatedMeansOfTransportation  Bicycle_1,
        relatedMeansOfTransportation  Pedestrian_1,
        relatedMeansOfTransportation  Tram_new_1,
        relatedMeansOfTransportation  Motorcycle_1,
        relatedMeansOfTransportation  Van_1,
        relatedMeansOfTransportation  Taxi_1


Individual: Truck_1

    Types:
        Truck
    Facts:
        cost_per_kilometer  0.2f


Individual: Tram_old_1

    Types:
        Tram_old
    Facts:
        cost_per_kilometer  0.65f


Individual: EUDriver_B_1

    Types:
        EUDriver_B


Individual: Bicycle_1

    Types:
        Bicycle
    Facts:
        cost_per_kilometer  0.0f


Individual: Medium_Luggage_Restriction_1

    Types:
        Medium_Luggage_Restriction
```

```

Individual: Escort_for_disabled_persons_1

    Types:
        Escort_for_disabled_persons


Individual: Motorcycle_1

    Types:
        Motorcycle
    Facts:
        cost_per_kilometer  0.06f


Individual: Taxi_1

    Types:
        Taxi
    Facts:
        cost_per_kilometer_additionalBasicPrice  2.1f,
        cost_per_kilometer  1.5f


Individual: BlindPerson_1

    Types:
        BlindPerson


Individual: Small_Luggage_Restriction_1

    Types:
        Small_Luggage_Restriction


Individual: Young_children_1

    Types:
        Young_children


Individual: S_Train_1

    Types:
        S_Train
    Facts:
        cost_per_kilometer  0.65f


Individual: PrivateCar_1

    Types:
        PrivateCar
    Facts:
        cost_per_kilometer  0.12f,
        model_name  "BMW 320i"


Individual: Transportee_Possibilities_User_3

    Types:
        Transportee_Possibilities
    Facts:
        owns_private_transp_vehicles  PrivateCar_1


Individual: Extralarge_Luggage_Restriction_1

    Types:
        Extralarge_Luggage_Restriction


Individual: Tram_new_1

    Types:
        Tram_new
    Facts:
        cost_per_kilometer  0.65f


Individual: Male

    Types:
        Gender


Individual: Female

    Types:
        Gender


Individual: User_3

    Types:
        Transportee
    Facts:
        relatedMeansOfTransportation  Truck_1,
        relatedMeansOfTransportation  Tram_old_1,
        relatedMeansOfTransportation  S_Train_1,
        relatedMeansOfTransportation  Bus_1,
        relatedMeansOfTransportation  PrivateCar_1,
        relatedMeansOfTransportation  U_Train_1,
        relatedMeansOfTransportation  Pedestrian_1,
        relatedMeansOfTransportation  Bicycle_1,
        relatedMeansOfTransportation  Tram_new_1,
        relatedMeansOfTransportation  Motorcycle_1,
        relatedMeansOfTransportation  Van_1,
        relatedMeansOfTransportation  Taxi_1,
        hasRestrictions  Wheelchair_1


Individual: Animal_Restriction_1

    Types:
        Animal_Restriction


Individual: Bus_1

    Types:
        Bus
    Facts:
        cost_per_kilometer  0.65f
```

```

Individual: Transportation_Preferences_User_2_u_train

    Types:
        Transportation_Preferences
    Facts:
        transportation_preference  2.0f,
        transportation_type  "U_Train"


Individual: U_Train_1

    Types:
        U_Train
    Facts:
        cost_per_kilometer  0.65f


Individual: Wheelchair_1

    Types:
        Wheelchair


Individual: Pedestrian_1

    Types:
        Pedestrian
    Facts:
        cost_per_kilometer  0.0f


Individual: Van_1

    Types:
        Van
    Facts:
        cost_per_kilometer  0.15f


Individual: Large_Luggage_Restriction_1

    Types:
        Large_Luggage_Restriction
```

BIBLIOGRAPHY

[AAAS94] Pankaj K. Agarwal, Noga Alon, Boris Aronov, and Subhash Suri. Can visibility graphs be represented compactly? *Discrete & Computational Geometry*, 12:347–365, 1994. (Cited on page 16.)

[AL01] Natasha Alechina and Brian Logan. State Space Search with Prioritised Soft Constraints. *Appl. Intell.*, 14(3):263–272, 2001. (Cited on page 25.)

[AW00] Rakesh Agrawal and Edward L. Wimmers. A Framework for Expressing and Combining Preferences. In Weidong Chen, Jeffrey F. Naughton, and Philip A. Bernstein, editors, *SIGMOD Conference*, pages 297–306. ACM, 2000. (Cited on page 25.)

[BCP03] Ronald Breiger, Kathleen M. Carley, and Philippa Pattison. Dynamic Social Network Modelling and Analysis: Workshop Summary and Papers. *J. Artificial Societies and Social Simulation*, 6(4), 2003. (Cited on page 30.)

[BD05] Christian Becker and Frank Dürr. On location models for ubiquitous computing. *Personal Ubiquitous Comput.*, 9(1):20–31, January 2005. (Cited on page 108.)

[Bel58] Richard Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958. (Cited on page 22.)

[BH08] Martin Brösamle and Christoph Hölscher. Architects seeing through the eyes of building users. In *Spatial Cognition in Architectural Design*, 2007. `http://www.sfbtr8.spatial-cognition.de/SCAD/` (last access on August 6, 2008.). (Cited on page 163.)

[BHHKM04] Ben-Moshe Boaz, Olaf Hall-Holt, Matthew J. Katz, and Joseph S. B. Mitchell. Computing the visibility graph of points within a polygon. In *SCG '04: Proceedings of the twentieth annual symposium on Computational geometry*, pages 27–35, New York, NY, USA, 2004. ACM. (Cited on page 16.)

[Bit01] Thomas Bittner. The Qualitative Structure of Built Environments. In *Fundamenta Informaticae*, volume 46 (nr. 1–2), pages 97–128, 2001. (Cited on pages 112, 117, and 163.)

[BKLM08] Thomas Barkowsky, Markus Knauff, Gérard Ligozat, and Daniel R. Montello, editors. *Spatial Cognition V: Reasoning, Action, Interaction, International Conference*

Spatial Cognition 2006, Bremen, Germany, September 24-28, 2006, Revised Selected Papers, volume 4387 of *Lecture Notes in Computer Science*. Springer, 2008. (Cited on pages 209 and 211.)

[BLOR05] François Bry, Bernhard Lorenz, Hans Jürgen Ohlbach, and Mike Rosner. A Geospatial World Model for the Semantic Web. In François Fages and Sylvain Soliman, editors, *PPSWR*, volume 3703 of *Lecture Notes in Computer Science*, pages 145–159. Springer, 2005. (Cited on page 159.)

[BM05] Juan Carlos Peris Broch and María Teresa Escrig Monferrer. Cognitive Maps for Mobile Robot Navigation: A Hybrid Representation Using Reference Systems. In *Proceedings of the 19th International Workshop on Qualitative Reasoning*, pages 179–186, 2005. (Cited on pages vii and 6.)

[BMR02] Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. Soft Constraint Logic Programming and Generalized Shortest Path Problems. *J. Heuristics*, 8(1):25–41, 2002. (Cited on page 25.)

[BP00] Paramvir Bahl and Venkata N. Padmanabhan. RADAR: An In-Building RF-Based User Location and Tracking System. In *Proceedings INFOCOM*, pages 775–784, 2000. (Cited on page 4.)

[CA04] Daniel Cagigas and Julio Abascal. Hierarchical Path Search with Partial Materialization of Costs for a Smart Wheelchair. *J. Intell. Robotics Syst.*, 39(4):409–431, 2004. (Cited on page 161.)

[CB00] Howie Choset and Joel Burdick. Sensor-Based Exploration: The Hierarchical Generalized Voronoi Graph. *International Journal of Robotics Research*, 19(2):96 – 125, February 2000. (Cited on pages 4 and 16.)

[CBGG97] Anthony G. Cohn, Brandon Bennett, John Gooday, and Nicholas Mark Gotts. Qualitative Spatial Representation and Reasoning with the Region Connection Calculus. In *GeoInformatica*, volume 1 (nr. 3), pages 275–316, 1997. (Cited on pages ix and 162.)

[CH01] Anthony G. Cohn and Shyamanta M. Hazarika. Qualitative Spatial Representation and Reasoning: An Overview. *Fundam. Inform.*, 46(1-2):1–29, 2001. (Cited on page 13.)

[CT05] David Caduff and Sabine Timpf. The Landmark Spider: Representing Landmark Knowledge for Wayfinding Tasks. In *Reasoning with Mental and External Diagrams: Computational Modeling and Spatial Assistance*, pages 30–35. AAAI Press. Stanford, 2005. (Cited on page 124.)

[CW01] Barbara Corona and Stephan Winter. Datasets for pedestrian navigation services. In J. Strobl, T. Blaschke, and G. Griesebner, editors, *In Angewandte Geographische Informationsverarbeitung XIII. Proceedings of the AGIT Symposium, Salzburg, Austria*, pages 84–89, 2001. (Cited on pages 4 and 162.)

[dBvKOS00] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. Visibility Graphs: Finding the Shortest Route. In *Computational Geometry: Algorithms and Applications*, chapter 15, pages 307–317. Springer, 2000. (Cited on pages 4 and 15.)

[Dey01] Anind K. Dey. Understanding and using context. *Personal Ubiquitous Comput.*, 5(1):4–7, 2001. (Cited on page 19.)

[DGP03] Robert Dale, Sabine Geldof, and Jean-Philippe Prost. Using Natural Language Generation for Navigational Assistance. In Michael J. Oudshoorn, editor, *ACSC*, volume 16 of *CRPIT*, pages 35–44. Australian Computer Society, 2003. (Cited on pages 13, 81, and 166.)

[DGP05] Robert Dale, Sabine Geldof, and Jean-Philippe Prost. Using Natural Language Generation in Automatic Route Description. *Journal of Research and Practice in Information Technology*, 37(1), 2005. (Cited on pages 4 and 166.)

[Dia79] Robert B. Dial. A model and algorithm for multicriteria route-mode choice. *Transportation Research*, 13B:311–316, February 1979. (Cited on pages 22 and 23.)

[Die00] Reinhard Diestel. *Graphentheorie*. Springer, ISBN 3-540-67656-2, 2000. (Cited on page 85.)

[Dij59] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959. (Cited on pages 22 and 40.)

[DK03] Matt Duckham and Lars Kulik. "Simplest" Paths: Automated Route Selection for Navigation. In Kuhn et al. [KWT03], pages 169–185. (Cited on page 23.)

[Dou04] Paul Dourish. What we talk about when we talk about context. *Personal and Ubiquitous Computing*, 8(1):19–30, 2004. (Cited on page 19.)

[Dur07] Pelin Dursun. Space Syntax in Architectural Design. In Ayse Sema Kubat, editor, *Proceedings of the 6th International Space Syntax Symposium (Istanbul)*, pages 056/01–56/12, 2007. (Cited on pages ix and 164.)

[Ead96] Peter Eades. Graph Drawing Methods. In Peter W. Eklund, Gerard Ellis, and Graham Mann, editors, *ICCS*, volume 1115 of *Lecture Notes in Computer Science*, pages 40–49. Springer, 1996. (Cited on page 31.)

[EB04] Birgit Elias and C Brenner. Automatic generation and application of landmarks in navigation data sets. In *Proceedings of the 11th International Symposium on Spatial Data Handling*, pages 469–480. Springer. Berlin, 2004. (Cited on pages 4 and 124.)

[EF91] Max J. Egenhofer and Robert D. Franzosa. Point Set Topological Relations. *International Journal of Geographical Information Systems*, 5:161–174, 1991. (Cited on pages 88 and 162.)

[EF96] Peter Eades and Qing-Wen Feng. Multilevel Visualization of Clustered Graphs. In Stephen C. North, editor, *Graph Drawing*, volume 1190 of *Lecture Notes in Computer Science*, pages 101–112. Springer, 1996. (Cited on page 31.)

[Eli03] Birgit Elias. Extracting Landmarks with Data Mining Methods. In Kuhn et al. [KWT03], pages 375–389. (Cited on page 124.)

[Eli07] Birgit Elias. Pedestrian Navigation - Creating a tailored geodatabase for routing. In *4th Workshop on Positioning, Navigation and Communication (WPNC '07), Hannover, Germany* , pages 41–47, 2007. (Cited on page 162.)

[FCE95] Qing-Wen Feng, Robert F. Cohen, and Peter Eades. Planarity for Clustered Graphs. In Paul G. Spirakis, editor, *ESA*, volume 979 of *Lecture Notes in Computer Science*, pages 213–226. Springer, 1995. (Cited on pages 32 and 37.)

[FF95] Carlos M. Fonseca and Peter J. Fleming. An Overview of Evolutionary Algorithms in Multiobjective Optimization. *Evolutionary Computation*, 3(1):1–16, 1995. (Cited on page 24.)

[FKKB+04] Christian Freksa, Markus Knauff, Bernd Krieg-Brückner, Bernhard Nebel, and Thomas Barkowsky, editors. *Spatial Cognition IV: Reasoning, Action, Interaction, International Conference Spatial Cognition 2004, Frauenchiemsee, Germany, October 11-13, 2004, RevisedSelected Papers*, volume 3343 of *Lecture Notes in Computer Science*. Springer, 2004. (Cited on pages 209 and 212.)

[FMW05] Gerald Franz, Hanspeter Mallot, and Jan Wiener. Graph-based Models of Space in Architecture and Cognitive Science - a Comparative Analysis. In *Proceedings of the 17th International Conference on Systems Research, Informatics and Cybernetics*, pages 30–38, 2005. (Cited on pages 5 and 11.)

[Fra03] Andrew U. Frank. Pragmatic Information Content - How to Measure the Information in a Route Description.

In Matt Duckham, M. Goodchild, and M. F. Worboys, editors, *Perspectives on Geographic Information Science (Chap. 4)*, pages 47–68. Taylor & Francis, 2003. (Cited on page 140.)

[Fre77] Linton C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40(1):35–41, March 1977. (Cited on page 164.)

[FT87] Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987. (Cited on pages 22 and 40.)

[Gab00] Harold N. Gabow. Path-based depth-first search for strong and biconnected components. *Information Processing Letters*, 74(3-4):107–114, 2000. (Cited on page 128.)

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison Wesley, Reading, MA, 1995. (Cited on pages 26 and 59.)

[GM91] Subir Kumar Ghosh and David M. Mount. An Output-Sensitive Algorithm for Computing Visibility Graphs. *SIAM J. Comput.*, 20(5):888–910, 1991. (Cited on page 19.)

[GM03] Pierre-Yves Gilliéron and Bertrand Merminod. Personal Navigation System for Indoor Applications. In *Proceedings of the 11th IAIN World Congress on Smart Navigation, Systems and Services*, Berlin, 2003. (Cited on pages 4, 10, and 165.)

[GNSW06] Michael Grossniklaus, Moira C. Norrie, Beat Signer, and Nadir Weibel. Putting Location-Based Services on the Map. In James D. Carswell and Taro Tezuka, editors, *W2GIS*, volume 4295 of *Lecture Notes in Computer Science*, pages 1–11. Springer, 2006. (Cited on page 134.)

[Gol95a] Reginald G. Golledge. Defining the criteria used in path selection. Technical Report UCTC No. 78, University of California, Transportation Center, 1995. (Cited on page 25.)

[Gol95b] Reginald G. Golledge. Path Selection and Route Preference in Human Navigation: A Progress Report. In *COSIT*, pages 207–222, 1995. (Cited on page 25.)

[Gol99] Reginald G. Golledge. Human Wayfinding and Cognitive Maps. In *Wayfinding behavior: Cognitive mapping and other spatial processes*, pages 5–45. Baltimore: Johns Hopkins University Press, 1999. (Cited on pages 113, 114, and 163.)

[Han79] Pierre Hansen. Bicriteria path problems. In *Multiple Criteria Decision Making Theory and Applications*, volume 177 of *Lecture Notes in Economics and Mathematical Systems*, pages 109–127. Springer, 1979. (Cited on pages 23 and 24.)

[HCF95] Daniel Hernández, Eliseo Clementini, and Paolino Di Felice. Qualitative Distances. In Andrew U. Frank and Werner Kuhn, editors, *Conference on Spatial Information Theory - COSIT*, volume 988 of *Lecture Notes in Computer Science*, pages 45–57. Springer, 1995. (Cited on page 13.)

[Hec05] Dominik Heckmann. *Ubiquitous User Modeling*. PhD thesis, Department of Computer Science, Saarland University, Germany, November 2005. (Cited on page 21.)

[HEH03] Michael D. Hendricks, Max J. Egenhofer, and Kathleen Hornsby. Structuring a Wayfinder's Dynamic Space-Time Environment. In Kuhn et al. [KWT03], pages 75–92. (Cited on pages 13 and 165.)

[HJ85] Stephen C. Hirtle and J. Jonides. Evidence of hierarchies in cognitive maps. *Memory and Cognition*, 13(3):208–217, 1985. (Cited on pages 31 and 163.)

[HJR00] YW Huang, N Jing, and E Rundensteiner. Optimizing Path Query Performance: Graph Clustering Strategies. *Transportation Research C*, 8(1–6):381–408, 2000. (Cited on page 159.)

[HKR06] Stefan Hansen, Alexander Klippel, and Kai-Florian Richter. Cognitive OpenLS Specification. Technical Report 012-10, University of Bremen, SFB/TR 8 Spatial Cognition, October 2006. (Cited on pages 53, 141, and 145.)

[HL04] Haibo Hu and Dik Lun Lee. Semantic Location Modeling for Location Navigation in Mobile Environment. In *Mobile Data Management*, pages 52–61. IEEE Computer Society, 2004. (Cited on pages 86 and 161.)

[HMV$^+$04] Christoph Hölscher, T Meilinger, G Vrachliotis, M Broesamle, and M Knauff. Finding the Way Inside: Linking Architectural Design Analysis and Cognitive Processes. In C Freksa, M Knauff, B Krieg-Brückner, B Nebel, and T Barkowsky, editors, *Spatial Cognition IV*, volume 3343 of *LNCS*, pages 1–23. Springer, 2004. (Cited on page 163.)

[HMV$^+$06] Christoph Hölscher, T Meilinger, G Vrachliotis, M Broesamle, and M Knauff. Up the down staircase: Wayfinding strategies in multi-level buildings. *Journal of Environmental Psychology*, 26(4):284–299, 2006. (Cited on page 163.)

[HMZM96] Robert C. Holte, T. Mkadmi, Robert M. Zimmer, and Alan J. MacDonald. Speeding up Problem Solving by Abstraction: A Graph Oriented Approach. *Artif. Intell.*, 85(1-2):321–361, 1996. (Cited on pages 42, 50, and 159.)

[HNR68] P. E. Hart, N. J. Nilsson, and B. Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968. (Cited on pages 43 and 139.)

[Hoc04] Hartwig Hochmair. Towards a classification of route selection criteria for route planning tools. In P. Fisher, editor, *Developments in Spatial Data Handling*, pages 481–492, Berlin, 2004. Springer. (Cited on page 25.)

[Hoc08] Hartwig Hochmair. Grouping of Optimized Pedestrian Routes for Multi-Modal Route Planning: A Comparison of Two Cities. In *Proceedings of the 11th AGILE International Conference on Geographic Information Science (AGILE 2008)*, LNGC, pages 339–358. Springer, 2008. (Cited on page 61.)

[HPZM96] Robert C. Holte, M. B. Perez, Robert M. Zimmer, and Alan J. MacDonald. Hierarchical A*: Searching Abstraction Hierarchies Efficiently. In *AAAI/IAAI, Vol. 1*, pages 530–535, 1996. (Cited on page 43.)

[HR02] Hartwig Hochmair and Martin Raubal. Topologic and Metric Decision Criteria for Wayfinding in the Real World and the WWW. In *Int. Symposium on Spatial Data Handling, SDH'02*, Ottawa, Canada, 2002. (Cited on page 25.)

[HRK06] Stefan Hansen, Kai-Florian Richter, and Alexander Klippel. Landmarks in OpenLS - A Data Structure for Cognitive Ergonomic Route Directions. In Raubal et al. [RMFG06], pages 128–144. (Cited on page 124.)

[HSB$^+$05] Dominik Heckmann, Tim Schwartz, Boris Brandherm, Michael Schmitz, and Margeritta von Wilamowitz-Moellendorff. GUMO - the General User Model Ontology. In *Proceedings of the 10th International Conference on User Modeling*, pages 428–432, Edinburgh, UK, 2005. LNAI 3538: Springer, Berlin Heidelberg. (Cited on page 21.)

[Ide06] Volker Iden. An Ontology of Persons and Vehicles for Movement in Geospatial Networks. Project Thesis, Institute for Informatics, University of Munich (LMU), 2006. (Cited on pages 57, 151, and 173.)

[Ide07] Volker Iden. A Language and a Framework for Rule-Based Modification of Semantic Models. Diplomarbeit / Diploma Thesis, Institute for Informatics, University of Munich (LMU), 2007. (Cited on pages 25, 58, 59, 151, and 173.)

[JF62] Lestor R. Ford Jr. and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, New Jersey, 1962. (Cited on page 22.)

[JHR96] N Jing, YW Huang, and E Rundensteiner. Hierarchical optimization of optimal path finding for transportation applications. In MT Özsu and K Barker, editors, *CIKM '96 Proceedings / Information and Knowledge Management*, pages 261–268. ACM, 1996. (Cited on page 159.)

[JHR98] Ning Jing, Yun-Wu Huang, and Elke A. Rundensteiner. Hierarchical Encoded Path Views for Path Query Processing: An Optimal Model and Its Performance Evaluation. *IEEE Trans. Knowl. Data Eng.*, 10(3):409–432, 1998. (Cited on pages 32, 34, 40, 43, 139, and 151.)

[KBG$^+$00] Benjamin Kuipers, Rob Browning, Bill Gribble, Mike Hewett, and Emilio Remolina. The spatial semantic hierarchy. *Artificial Intelligence*, 119:191–233, 2000. (Cited on page 161.)

[KBS06] Bernd Krieg-Brückner and Hui Shi. Orientation Calculi and Route Graphs: Towards Semantic Representations for Route Descriptions. In Raubal et al. [RMFG06], pages 234–250. (Cited on page 167.)

[KK77] Leonard Kleinrock and Farouk Kamoun. Hierarchical Routing for Large Networks; Performance Evaluation and Optimization. *Computer Networks*, 1:155–174, 1977. (Cited on page 29.)

[Kno91] Craig A. Knoblock. Search Reduction in Hierarchical Problem Solving. In *In Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 686–691, 1991. (Cited on page 159.)

[Kof35] Kurt Koffka. *Principles of Gestalt Psychology*. Lund Humphries, London, 1935. (Cited on page 119.)

[KTS03] Benjamin Kuipers, D Tecuci, and B Stankiewicz. The Skeleton in the Cognitive Map: A Computational and Empirical Exploration. *Environment and Behavior*, 35(1):80–106, 2003. (Cited on pages 123, 126, and 160.)

[KW05] Alexander Klippel and Stephan Winter. Structural salience of landmarks for route directions. In *Conference on Spatial Information Theory - COSIT*, pages 347–362. Springer, 2005. (Cited on page 124.)

[KWT03] Werner Kuhn, Michael F. Worboys, and Sabine Timpf, editors. *Spatial Information Theory. Foundations of Geographic Information Science, International Conference,* *COSIT 2003, Ittingen, Switzerland, September 24-28, 2003, Proceedings*, volume 2825 of *Lecture Notes in Computer Science*. Springer, 2003. (Cited on pages 199, 200, and 202.)

[LA04] Jyh-Ming Lien and Nancy M. Amato. Approximate convex decomposition of polygons. In *SCG '04: Proceedings of the twentieth annual symposium on Computational geometry*, pages 17–26, New York, NY, USA, 2004. ACM. (Cited on page 162.)

[Lat90] J. C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, MA, Decembre 1990. (Cited on pages 14 and 118.)

[LH03] Sylvain Lefebvre and Samuel Hornus. Automatic Cell-and-Portal Decomposition. Technical Report 4898, INRIA, July 2003. (Cited on page 162.)

[Liu96] Bing Liu. Intelligent route finding: Combining knowledge, cases and an efficient search algorithm. In *Proceedings of the 12th European Conference on Artificial Intelligence*, pages 380–384, 1996. (Cited on pages 159 and 160.)

[LKLS05] Gary Look, Buddhika Kottahachchi, Robert Laddaga, and Howard E. Shrobe. A location representation for generating descriptive walking directions. In Robert St. Amant, John Riedl, and Anthony Jameson, editors, *IUI*, pages 122–129. ACM, 2005. (Cited on pages 117 and 167.)

[LL08] Dandan Li and Dik Lun Lee. A Lattice-Based Semantic Location Model for Indoor Navigation. In Xiaofeng Meng, Hui Lei, Stéphane Grumbach, and Hong Va Leong, editors, *MDM*, pages 17–24. IEEE, 2008. (Cited on page 161.)

[LOS06] Bernhard Lorenz, Hans Jürgen Ohlbach, and Edgar-Philipp Stoffel. A Hybrid Spatial Model for Representing Indoor Environments. In *Proceedings of the 6th International Symposium on Web and Wireless Geographical Information Systems*, volume 4295 of *LNCS*, pages 102–112. Springer, 2006. (Cited on page 87.)

[Lou83] Ronald Prescott Loui. Optimal Paths in Graphs with Stochastic or Multidimensional Weights. *Commun. ACM*, 26(9):670–676, 1983. (Cited on page 22.)

[LSM98] Josep Lladós, Gemma Sánchez, and Enric Martí. A String Based Method to Recognize Symbols and Structural Textures in Architectural Plans. In *GREC '97: Selected Papers from the Second International Workshop on Graphics Recognition, Algorithms and Systems*, pages 91–103, London, UK, 1998. Springer-Verlag. (Cited on page 167.)

[LTK02] Paul U. Lee, Heike Tappe, and Alexander Klippel. Acquisition of Landmark Knowledge from Static and Dynamic Presentation of Route Maps. *KI*, 16(4):32–34, 2002. (Cited on page 124.)

[Lyn60] Kevin Lynch. *The Image of the City*. The MIT Press, June 1960. (Cited on pages 113, 114, and 163.)

[MdlC05] Lawrence Mandow and José-Luis Pérez de-la Cruz. A New Approach to Multiobjective A* Search. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *IJCAI*, pages 218–223. Professional Book Center, 2005. (Cited on page 24.)

[Miz04] Doreen Mizzi. A Mobile Navigation Assistance System Using Natural Language Generation. Diploma thesis, Dept. of Computer Science and AI, University of Malta, 2004. (Cited on pages 13, 14, and 166.)

[Mon93] Daniel R. Montello. Scale and Multiple Psychologies of Space. In *COSIT*, pages 312–321, 1993. (Cited on pages 118, 119, and 161.)

[MP08] Dejian Meng and Stefan Poslad. A reflective context-aware system for spatial routing applications. In *MPAC '08: Proceedings of the 6th international workshop on Middleware for pervasive and ad-hoc computing*, pages 54–59, New York, NY, USA, 2008. ACM. (Cited on page 22.)

[MR96] Dario Maio and Stefano Rizzi. Layered Knowledge Architecture For Navigation-Oriented Environment Representation. Technical Report CIOC-C.N.R. no. 108, University of Bologna, 1996. (Cited on page 161.)

[MS01] Harvey J. Miller and Shih-Lung Shaw. *Geographic Information Systems for Transportation: Principles and Applications*. Oxford University Press, 2001. (Cited on page 27.)

[MS07] Stefan Münzer and Christoph Stahl. Providing individual route instructions for indoor wayfinding in complex, multi-level buildings. In F. Probst and C. Keßler, editors, *Proceedings of the 5th Geographic Information Days, Münster*, pages 241–246. IfGIprints, 2007. (Cited on pages ix, 165, and 166.)

[MSK06] Matt MacMahon, Brian Stankiewicz, and Benjamin Kuipers. Walk the Talk: Connecting Language, Knowledge, and Action in Route Instructions. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference - AAAI*. AAAI Press, 2006. (Cited on pages 14 and 167.)

[MW06] Peter Mooney and Adam C. Winstanley. An evolutionary algorithm for multicriteria path optimization problems. *International Journal of Geographical Information Science*, 20(4):401–423, 2006. (Cited on page 24.)

[Nas01] Hugo A. D. Do Nascimento. A framework for human-computer interaction in directed graph drawing. In *In the Proceedings of the Australian Symposium on Information Visualisation*, pages 63–69. Australian Computer Society, Inc, 2001. (Cited on page 31.)

[Nil69] Nils J. Nilsson. A mobile automaton: An application of artificial intelligence techniques. In *IJCAI*, pages 509–520, 1969. (Cited on page 15.)

[Ope] OGC OpenLS specification `http://www.opengeospatial.org/standards/ols` (last accessed on 9th October 2008). (Cited on pages vii, 52, 53, and 159.)

[ORLS06] Hans Jürgen Ohlbach, Mike Rosner, Bernhard Lorenz, and Edgar-Philipp Stoffel. NL Navigation Commands from Indoor WLAN fingerprinting position data. REWERSE Deliverable A1-D7, Institute for Informatics, Ludwig-Maximilians-Universität München, 2006. (Cited on pages 4, 12, and 135.)

[OS08] Hans Jürgen Ohlbach and Edgar-Philipp Stoffel. Versatile Route Descriptions for Pedestrian Guidance in Buildings – Conceptual Model and Systematic Method. In *AGILE '08 Online Proceedings*, 2008. (Cited on pages 118 and 139.)

[ÓSY87] Colm Ó'Dúnlaing, Micha Sharir, and Chee-Keng Yap. Generalized Voronoi Diagrams for a Ladder: II. Efficient Construction of the Diagram. *Algorithmica*, 2:27–59, 1987. (Cited on pages 16 and 19.)

[Pas84] Romedi Passini. Spatial Representations, a Wayfinding Perspective. *Environmental Psychology*, 4(2):153–164, 1984. (Cited on pages 14, 123, and 163.)

[Pas96] Romedi Passini. Wayfinding design: logic, application and some thoughts on universality. *Design Studies*, 17(3):319–331, 1996. (Cited on pages 14 and 163.)

[Pep97] John Peponis. Geometries of Architectural Description: shape and spatial configuration. In *Space Syntax, First International Symposium Proceedings*, pages 33.1–33.7, 1997. (Cited on page 164.)

[PG96] Lutz Plümer and Gerhard Gröger. Nested Maps – a Formal, Provably Correct Object Model for Spatial Aggregates. In *Proceedings of the 4th ACM International Workshop on Advances in Geographic Information Systems*, pages 76–83. ACM Press, 1996. (Cited on pages 87 and 161.)

[PKH$^+$05] Perttu Prusi, Anssi Kainulainen, Jaakko Hakulinen, Markku Turunen, Esa-Pekka Salonen, and Leena Helin. Towards Generic Spatial Object Model and Route Guidance Grammar for Speech-based Systems. In *INTERSPEECH-2005, Proceedings of the 9th European Conference on Speech Communication and Technology. Lisbon, Portugal*, pages 1917–1920. ISCA Archive, 2005. (Cited on page 166.)

[PL94] Alexandra Poulovassilis and Mark Levene. A nested-graph model for the representation and manipulation of complex objects. *ACM Transactions on Information Systems*, 12:35–68, 1994. (Cited on pages 32, 37, and 38.)

[Pri05] Nissanka B. Priyantha. *The Cricket Indoor Location System*. PhD thesis, Massachusetts Institute of Technology, 2005. (Cited on page 4.)

[PS97] Stefano Pallottino and Maria Grazia Scutellà. Shortest Path Algorithms in Transportation models: classical and innovative aspects. Technical Report TR-97-06, University of Pisa, 1997. (Cited on pages 22 and 23.)

[PSN08] Ingmar Posner, Derik Schröter, and Paul M. Newman. Online generation of scene descriptions in urban environments. *Robotics and Autonomous Systems*, 56(11):901–914, 2008. (Cited on page 167.)

[PSZ06] Christine Parent, Stefano Spaccapietra, , and Esteban Zimányi. *Conceptual Modeling for Traditional and Spatio-Temporal Applications: The MADS Approach*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. (Cited on page 159.)

[PWR$^+$97] John Peponis, J. Wineman, M. Rashid, S.H. Kim, and S. Bafna. On the description of shape and spatial configuration inside buildings: convex partitions and their local properties. *Environment and Planning B: Planning and Design*, 24(5):761–781, 1997. (Cited on page 164.)

[PZC90] John Peponis, C Zimring, and YK Choi. Finding the Building in Wayfinding. *Environment and Behavior*, 22(5):555–590, 1990. (Cited on page 164.)

[Rai04] Marcus Raitner. Visual Navigation of Compound Graphs. In János Pach, editor, *Graph Drawing*, volume 3383 of *Lecture Notes in Computer Science*, pages 403–413. Springer, 2004. (Cited on pages 31 and 34.)

[Rau01] Martin Raubal. Ontology and epistemology for agent-based wayfinding simulation. *International Journal of Geographical Information Science*, 15(7):653–665, 2001. (Cited on page 13.)

[Ric07] Kai-Florian Richter. A Uniform Handling of Different Landmark Types in Route Directions. In Stephan Winter, Matt Duckham, Lars Kulik, and Benjamin Kuipers, editors, *COSIT*, volume 4736 of *Lecture Notes in Computer Science*, pages 373–389. Springer, 2007. (Cited on page 167.)

[RJB04] James Rumbaugh, Ivar Jacobson, and Grady Booch. *Unified Modeling Language Reference Manual, The (2nd Edition)*. Pearson Higher Education, 2004. (Cited on page 30.)

[RK04] Kai-Florian Richter and Alexander Klippel. A Model for Context-Specific Route Directions. In Freksa et al. [FKKB$^+$04], pages 58–78. (Cited on pages 124, 141, and 167.)

[RK06] Kai-Florian Richter and Alexander Klippel. Before or After: Prepositions in Spatially Constrained Systems. In Barkowsky et al. [BKLM08], pages 453–469. (Cited on page 167.)

[RM04] Mike Rosner and Doreen Mizzi. Three Scenarios for Navigational Advice Generation. In Anja Belz, Roger Evans, and Paul Piwek, editors, *INLG04 Posters: Extended abstracts of posters presented at the Third International Conference on Natural Language Generation*, pages 36–40, 2004. (Cited on pages 4 and 166.)

[RMFG06] Martin Raubal, Harvey J. Miller, Andrew U. Frank, and Michael F. Goodchild, editors. *Geographic Information Science, 4th International Conference, GIScience 2006, Münster, Germany, September 20-23, 2006, Proceedings*, volume 4197 of *Lecture Notes in Computer Science*. Springer, 2006. (Cited on pages 203 and 204.)

[RsMMSB05] Axel Rottmann, Óscar Martínez Mozos, Cyrill Stachniss, and Wolfram Burgard. Semantic Place Classification of Indoor Environments With Mobile Robots using Boosting. In *in Proc. of the National Conference on Artificial Intelligence (AAAI*, pages 1306–1311, 2005. (Cited on page 167.)

[RT04] Urs-Jakob Rüetschi and Sabine Timpf. Modelling Wayfinding in Public Transport: Network Space and Scene Space. In Freksa et al. [FKKB$^+$04], pages 24–41. (Cited on page 11.)

[RT05] Urs-Jakob Rüetschi and Sabine Timpf. Using Image Schemata to Represent Meaningful Spatial Configurations. In *OTM Workshops*, volume 3762 of *LNCS*, pages 1047–1055. Springer, 2005. (Cited on page 86.)

[RW99] Martin Raubal and Michael Worboys. A Formal Model of the Process of Wayfinding in Built Environments.

In *Proceedings of the International Conference on Spatial Information Theory*, volume 1661 of *LNCS*, pages 381–401. Springer, 1999. (Cited on pages 4 and 163.)

[RW02] Martin Raubal and Stephan Winter. Enriching Wayfinding Instructions with Local Landmarks. In Max J. Egenhofer and David M. Mark, editors, *GIScience*, volume 2478 of *Lecture Notes in Computer Science*, pages 243–259. Springer, 2002. (Cited on pages 4 and 124.)

[Sab66] Gert Sabidussi. The centrality index of a graph. *Psychometrika*, 31(4):581–603, December 1966. (Cited on page 164.)

[Sac73] Earl D. Sacerdoti. Planning in a Hierarchy of Abstraction Spaces. In *IJCAI*, pages 412–422, 1973. (Cited on pages 41, 50, 81, 123, and 159.)

[SCCW91] Bradley S. Stewart and III Chelsea C. White. Multiobjective A*. *Journal of the ACM*, 38(4):775–814, 1991. (Cited on page 24.)

[Sch06] Matthias Schmeisser. PlanML: A Markup Language for Navigational Planning. Diplomarbeit / Diploma Thesis, Institute for Informatics, University of Munich (LMU), 2006. (Cited on pages 145 and 148.)

[SFG97] Shashi Shekhar, Andrew Fetterer, and Bjajesh Goyal. Materialization Trade-Offs in Hierarchical Shortest Path Algorithms. In Michel Scholl and Agnès Voisard, editors, *SSD*, volume 1262 of *Lecture Notes in Computer Science*, pages 94–111. Springer, 1997. (Cited on pages 40, 43, and 159.)

[SH06] Christoph Stahl and Jens Haupert. Taking Location Modelling to New Levels: A Map Modelling Toolkit for Intelligent Environments. In Mike Hazas, John Krumm, and Thomas Strang, editors, *LoCA*, volume 3987 of *Lecture Notes in Computer Science*, pages 74–85. Springer, 2006. (Cited on pages 84, 152, 165, and 171.)

[SLO07] Edgar-Philipp Stoffel, Bernhard Lorenz, and Hans Jürgen Ohlbach. Towards a Semantic Spatial Model for Pedestrian Indoor Navigation. In JL Hainaut and E Rundensteiner, editors, *ER '07 Workshops / Advances in Conceptual Modeling*, volume 4802 of *LNCS*, pages 328–337. Springer, 2007. (Cited on page 118.)

[SM95] Kozo Sugiyama and Kazuo Misue. A Generic Compound Graph Visualizer/Manipulator: D-ABDUCTOR. In Franz-Josef Brandenburg, editor, *Graph Drawing*, volume 1027 of *Lecture Notes in Computer Science*, pages 500–503. Springer, 1995. (Cited on pages 32 and 37.)

[sMMTJ+07] Óscar Martínez Mozos, Rudolph Triebel, Patric Jensfelt, Axel Rottmann, and Wolfram Burgard. Supervised semantic labeling of places using information extracted from sensor data. *Robot. Auton. Syst.*, 55(5):391–402, 2007. (Cited on page 167.)

[SMR06] Hui Shi, Christian Mandel, and Robert J. Ross. Interpreting Route Instructions as Qualitative Spatial Actions. In Barkowsky et al. [BKLM08], pages 327–345. (Cited on pages 13 and 14.)

[SRK07] Annegret Stark, Marcel Riebeck, and Jürgen Kawalek. How to Design an Advanced Pedestrian Navigation System: Field Trial Results. . In *Proceedings of the Fourth IEEE Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications IDAACS September 6-8, 2007, Dortmund, Germany.*, pages 690–694, 2007. (Cited on page 162.)

[SSO08] Edgar-Philipp Stoffel, Korbinian Schoder, and Hans Jürgen Ohlbach. Applying Hierarchical Graphs to Pedestrian Indoor Navigation. In Walid G. Aref, Mohamed F. Mokbel, and Markus Schneider, editors, *GIS*, page 54. ACM, 2008. (Cited on pages 126 and 132.)

[Sta] Christoph Stahl. The Yamamoto Map Modelling Toolkit., `http://w5.cs.uni-sb.de/$\sim$stahl/yamamoto/index.html`. (Cited on pages ix, 165, 166, and 171.)

[Ste99] John G. Stell. Granulation for Graphs. In Christian Freksa and David M. Mark, editors, *COSIT*, volume 1661 of *Lecture Notes in Computer Science*, pages 417–432. Springer, 1999. (Cited on pages vii, 28, 32, 36, and 160.)

[TAK+05] Vassileios Tsetsos, Christos Anagnostopoulos, Panayiotis Kikiras, Tilemahos Hasiotis, and Stathes Hadjiefthymiades. A Human-centered Semantic Navigation System for Indoor Environments. In *Proceedings of the IEEE/ACS International Conference on Pervasive Services (ICPS'05)*, pages 146–155. IEEE Computer Society, 2005. (Cited on page 166.)

[TAKH06] Vassileios Tsetsos, Christos Anagnostopoulos, Panayotis Kikiras, and Stathes Hadjiefthymiades. Semantically Enriched Navigation for Indoor Environments. *International Journal of Web and Grid Services*, 2(4):453–478, 2006. (Cited on page 166.)

[Tar72] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972. (Cited on page 128.)

[TF97] Sabine Timpf and Andrew U. Frank. Using Hierarchical Spatial Data Structures for Hierarchical Spatial

Reasoning. In Stephen C. Hirtle and Andrew U. Frank, editors, *Spatial Information Theory: A Theoretical Basis for GIS, Proceedings of the International Conference COSIT '97*, volume 1329 of *Lecture Notes in Computer Science*, pages 69–83. Springer, 1997. (Cited on page 161.)

[Tom07] Martin Tomko. *Destination Descriptions in Urban Environments*. PhD thesis, Department of Geomatics, The University of Melbourne, 2007. (Cited on pages 81 and 160.)

[TP02] Alasdair Turner and Alan Penn. Encoding Natural Movement as an Agent-Based System: An Investigation into Human Pedestrian Behaviour in the Built Environment. *Environment and Planning B: Planning and Design*, 29(4):473–490, 2002. (Cited on pages 141 and 164.)

[Vin74] Philippe Vincke. Problèmes multicritères. *Cahiers du Centre d'Etudes de Recherche Opérationnelle*, 16:425–439, 1974. (Cited on page 23.)

[Vin92] Philippe Vincke. *Multicriteria Decision-Aid*. John Wiley & Sons, New York, Chichester, 1992. (Cited on page 23.)

[Voi03] Horatiu Voicu. Hierarchical cognitive maps. *Neural Networks*, 16(5-6):569–576, 2003. (Cited on page 163.)

[Wal04] Jan Oliver Wallgrün. Autonomous Construction of Hierarchical Voronoi-Based Route Graph Representations. In Freksa et al. [FKKB$^+$04], pages 413–433. (Cited on pages vii, 16, and 17.)

[Was07] Martin Wassermann. Konzeption und Realisierung eines Web basierten Mediatorsystems für Graphenalgorithmen über verteilte geographische Netze. Diplomarbeit / Diploma Thesis, Institute for Informatics, University of Munich (LMU), 2007. (Cited on pages xi, 52, and 55.)

[WBT07] Emily Whiting, Jonathan Battat, and Seth Teller. Topology of Urban Environments: Graph construction from multi-building floor plan data. In *Dong A, Moere VA, Gero JS (eds.) Computer-Aided Architectural Design Futures 2007. vol XII*, pages 115–128, 2007. (Cited on pages 162 and 166.)

[Wel85] Emo Welzl. Constructing the Visibility Graph for n-Line Segments in O(n^2) Time. *Inf. Process. Lett.*, 20(4):167–171, 1985. (Cited on page 19.)

[Wer23] Max Wertheimer. Laws of Organization in Perceptual Forms. *Psychologische Forschung*, 4:301–350, 1923. (Cited on page 119.)

[WFR+07] Jan M Wiener, Gerhard Franz, Nicole Rossmanith, Andreas Reichelt, Hanspeter Mallot, and Heinrich Bülthoff. Isovist analysis captures properties of space relevant for locomotion and experience. *Perception*, 36(7):1066–1083, 06 2007. (Cited on page 163.)

[Whi06] Emily Whiting. Geometric, Topological & Semantic Analysis of Multi-building Floor Plan Data. Master's thesis, Massachusetts Institute of Technology, Department of Architecture, May 2006. (Cited on pages vii, 16, 17, 162, and 166.)

[Win01] Stephan Winter. Weighting the Path Continuation in Route Planning. In Walid G. Aref, editor, *ACM-GIS*, pages 173–176. ACM, 2001. (Cited on page 23.)

[WM03a] Jan M. Wiener and Hanspeter A. Mallot. 'Fine-to-Coarse' Route Planning and Navigation in Regionalized Environments. *Spatial Cognition and Computation*, 3(4):331–358, Dez 2003. (Cited on pages 4 and 163.)

[WM03b] Jan M. Wiener and Hanspeter A. Mallot. Route Planning in Hierarchically Structured Environments: From Places to Regions. *EuroCogSci*, 2003. (Cited on page 163.)

[YWR03] Jing Yang, Matthew O. Ward, and Elke A. Rundensteiner. Hierarchical exploration of large multivariate data sets. In Frits H. Post, Gregory M. Nielson, and Georges-Pierre Bonneau, editors, *Data Visualization: The State of the Art*, pages 201–212. Kluwer, 2003. (Cited on page 31.)

[ZF96] Kai Zimmermann and Christian Freksa. Qualitative Spatial Reasoning Using Orientation, Distance, and Path Knowledge. *Appl. Intell.*, 6(1):49–58, 1996. (Cited on page 13.)

INDEX