Dissertation

Enhancing Isochrones in Multimodal Spatial Networks

Nikolaus Krismer

submitted to the Faculty of Mathematics, Computer Science and Physics of the University of Innsbruck

in partial fulfillment of the requirements for the degree of "Doctor of Philosophy"

Innsbruck, November 2017

Advisor: Univ.-Prof. Dr. Günther Specht

Abstract

Isochrones are an important tool for carrying out reachability analyses. In Multimodal Spatial Networks they are hard to compute, since multiple modalities and schedules of the public transportation systems need to be considered. Throughout this thesis new algorithms to compute isochrones in such networks are developed. Those include loading parts of the data source with the help of so called tile regions as well as computations on previously calculated results. The application capabilities of isochrones are extended by defining averaged and time-invariant isochrones. By introducing elevation aware isochrone computation the accuracy of the results is further increased. An approach that is able to tailor isochrones to individual users is presented as well.

One of the major problems in the field of isochrone computation is the acquisition of the necessary data. To address the lack of datasets on which such calculations can be performed, a generalized dataset creation is introduced that allows not only to create real-world models, but also to include information from external data sources. Data is merged with information about the public transportation system and with Digital Elevation Models that include elevation data. To ease the visualization of computed isochrones in real-world datasets a web-application called “IsoMap” is presented. It can be used to vary computation parameters and for basic analyses within the results.

Contents

CHAPTER 1

Introduction

1.1. Motivation

For more than one decade interactive maps play an important role in computer science. With the introduction of map services from big companies like Microsoft and Google in 2005, various applications building on them have become popular in the world wide web. Besides other, two important tasks are regularly carried out with these services: finding places and calculating routes between two locations A and B. The latter is known as routing. Although online maps are not the only or even first tool that enables routing computations, i.e. navigation devices by TomTom have entered the market in 2001, these services are the ones that allow fast routing computation for everybody who has access to the internet. Another task enabled with the help of interactive maps, is the calculation of reachabilities, which deals with the areas that can be reached within a certain time span from a query point (or vice versa dealing with the areas from which a query point can be reached within a time span).

Reachability plays an important role in various applications. It is important when planning where to spend spare time or when looking for a place to life. Governments need to take care of reachability during urban planning or

when designing public buildings like schools. For businesses such as public transportation operators it even is a key factor for success.

The foundation to every route finding and reachability analysis is the presence of geographical data. It is necessary to have access to the data itself in order to perform relevant queries. Advances in computer hardware allow to store terabytes of data even on personal systems. This makes it possible to model the whole world in detail and to store the gained information on a single server. As a result not only big companies like Apple, Google and Microsoft gather geographical data, but also non-commercial enterprises. One of the most utilized, analyzed and cited Volunteered Geographic Information (VGI)-platforms, with an increasing popularity over the past few years, is OpenStreetMap (OSM) [86]. This community project not only allows to share data about geographical realities, but also to use the gathered data under the Open Data Commons Open Database License (ODbL) [96]. The license states that the crowdsourced data is freely available without paying any additional fee as long as the authors are attributed and the adopted data is also provided under the same license, thus enabling scientists all over the world to work with geographical data.

The availability of geographical information allows for the implementation of algorithms that are tailored to real-world problems like facilitating optimized routing rather than routing in abstract networks. Therefore, algorithms that were developed in the 20th century, like Dijkstra's algorithm was in 1959 [23] and A-Star in 1968 [46], were further improved to Contraction Hierarchies (CH) in 2008 and Customized Route Planning (CRP) in 2011. When it comes to reachability analyses, algorithms have not been developed as quickly. Most of the current approaches rely on Dijkstra's algorithm and as soon as schedules of the public transportation system come into play, advanced algorithms are lacking. Hence, computation time becomes an issue. Therefore, one of the objectives of this thesis is to evaluate currently available algorithms for reachability calculation and to find ways to decrease the computation time needed besides extending isochrone application capabilities with elevation aware computation, averaged and time-invariant calculations and user-tailored isochrones.

Since there are multiple different types of reachability known, which are computed in a very similar way, the next section lists some of them and explains their difference, before research objectives are presented.

1.2. Different Types of Reachability Analyses

There exist multiple types of reachability analyses. At first, isodistances and isochrones need to be distinguished. They differ in the constraint that is used for limiting the computation.

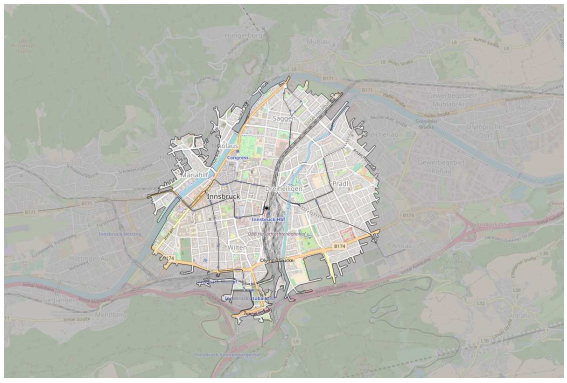


Figure 1.1.: 2000 meter isodistance.

Isodistances use a distance based approach to determine the area that is reachable. The isodistance itself is defined as the borderline between the reachable and the unreachable area. The resulting isodistance is either a circle if the information about the road network represented in the geographical data is neglected or a polygon determined by roads and ways such as the one shown in Figure 1.1, where the reachable area is plotted as brightened area, whereas the rest of the region is not reachable and shown as darkened area.

In contrast, isochrones use time as the limiting factor instead of distance, but are also defined as the borderline between a reachable area (determined by time rather than distance) and a not reachable area. Figure 1.2 shows an unimodal isochrone, where the isochrone itself is the black border between the darkened and the brightened area.

In a road network each way is assigned with additional information, e.g. with a speed limit, that needs to be taken into account when traveling. Even basic isochrones that allow movement within the data by only one means of transportation (so called unimodal), are more computational intense to compute than isodistances, since additional properties, namely the traveling speed and speed limits connected to every way

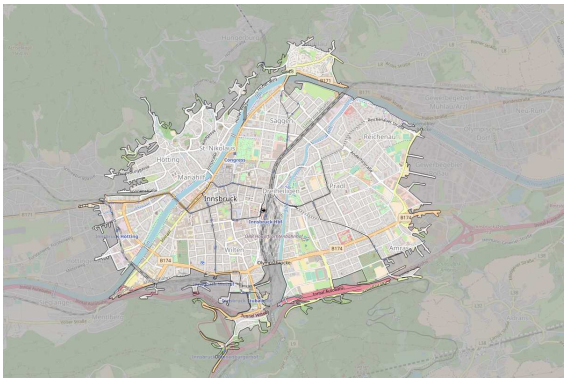


Figure 1.2.: Unimodal 10min isochrone using a travel speed of 20km/h.

(or way segment), have to be taken into account. If multiple transportation modalities (multimodal) can be used to travel around, things get even more complicated, e.g. when taking a car and walking by foot is allowed.

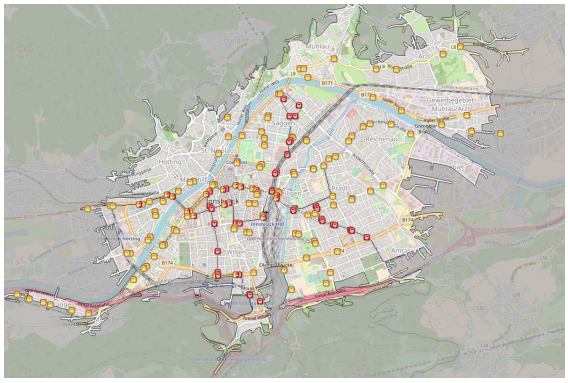


Figure 1.3.: Multimodal 10min isochrone using a travel speed of 20km/h.

Networks can be classified as continuous or discrete. Road networks are typically continuous in the space and time dimension. The edges represent streets while the vertices refer to junctions, signs or point of interests (POIs). When using the public transportation system schedules need to be considered. Since between two stops of a bus or train leaving the vehicle is not possible, the underlying data is discrete in space and time. A possible result of a multimodal isochrone within the city of Innsbruck, Austria, starting at the main station that also allows to travel by buses and trams of the public transportation system is shown in Figure 1.3.

When computing isochrones in a network that is continuous in space and time the term "Isochrone in a Continuous Network" is used. Besides the maximal travel duration also a travel speed needs to be known in contrast to isodistances. The term "Isochrone in a Multimodal Spatial Network" refers to the fact the the underlying network is partially discrete in space and time. When working in such a Multimodal Spatial Network (MMSN) the time and speed limitations are not sufficient to compute an isochrone. Since schedules are bound by arrival or departure time a starting date and time need to be supplied as well. In addition, the travel direction is also important now. A point of interest can be reachable from many places, while it may not be true that many places are reachable from the same POI. The fact that the network is discrete in space leads to the problem that the resulting isochrone might be disconnected. While an isochrone in an unimodal network is always a polygon (or a subclass, e.g. circles or squares) an isochrone in a MMSN can consist of multiple disconnected polygons, also known as multi-polygon.

To summarize, there are at least three different methods that can be used to perform a reachability analysis:

Isodistances are limited by a distance (typically given in meters) and are carried out in networks that are continuous in space and time.

Isochrones in Continuous Networks use time as a limiting factor and are calculated in networks that are continuous in space and time. In contrast to isodistances a maximal travel duration and a travel speed need to be known for the purpose of isochrone computation. Different costs on network edges, like speed limits, have to be taken into account to estimate travel times correctly.

Isochrones in Multimodal Spatial Networks allow for traveling in space and time-dependent networks. The underlying network is at least partially discrete in space and time. In addition to isochrones in continuous networks, a date and time and a travel direction (starting from or ending at the query point) need to be set. This is mandatory in order to take care of schedule based modes of travel like trains and buses. The resulting isochrone can consist of multiple, possibly disconnected polygons.

1.3. Research Objectives and Contributions

The main goal of this thesis is to deal with problems in the area of reachability analyses using isochrones. Four topics have been chosen to be addressed in detail. The first one deals with the fact that algorithms capable to calculate isochrones and the reachable area within an isochrone are computational intense. While other geographical queries like routing are highly optimized in terms of memory consumption and runtime, isochrones are not as easy to compute. Therefore, the performance of algorithms is going to be analyzed and improved. Several enhancements are made to the domain of isochrone calculation. Besides the definition of averaged and time-invariant isochrones, methods to tailor isochrones to individual users and computation based on known results are discussed. Another disadvantage is that there is no standardized way to create Multimodal Spatial Networks. Doing so, information from multiple data sources needs to be gathered and linked. This affects geographical data, e.g. adding missing information about the elevation of a point, as well as non-geographical data, e.g. a schedule of a train. For demonstration and display purposes and making reachability computation available to people, a user interface is developed and presented. In particular, the following research questions are investigated as part of the thesis at hand:

- How can algorithms capable of computing isochrones in Multimodal Spatial Networks be further improved?
- What issues arise when using a time span instead of a specific moment in time change the reachability of places?

- In which manner are isochrones influenced if elevation information is taken into account?
- To what extend is it possible to compute customized isochrones for individuals?
- How can the different types of data sources be merged into a common model in order to create a Multimodal Spatial Network?
- What could be an appropriate way to present isochrones in a visual and interactive fashion?

The contributions of this thesis can be summarized as follows:

Algorithm Improvements. Based on the ideas developed by Bauer et al. [9] and Gamper et al. [33, 34] various methods are evaluated and implemented to compute isochrones. The ideas developed by M. Innerebner [51], which include a hybrid data loading approach between the former two algorithms, are analyzed and implemented as part of this thesis. A newly developed way to load data from a spatial database in order to calculate isochrones is presented while general improvements are described and applied to all those algorithms. The runtime and memory consumption of all the presented algorithms are evaluated and compared to each other.

Isochrone Enhancements. Some aspects that are well known in the context of routing are still missing for isochrones. An obvious one when having an undulating landscape is elevation awareness. Depending on the traveling modality the difference in altitude is important. Hence, it should not be neglected that different speed adjustments have to be made. With the information about elevation at hand additional improvements can be made. One that has not been done before is to tailor isochrones to individual persons. The reachability when traveling by bike depends on how fast a cyclist can go and this is very different depending on several factors like age and personal fitness. Using these information a profile can be created that later is available during isochrone creation.

Another important issue when it comes to reachability, especially when using the public transportation system, is that it depends on the day of the week and the time of the day. While in the morning at 8 o'clock it might be quite fast to reach your working place by bus, it might be much slower when doing the same at midnight. To deal with this issue averaged isochrones are defined that aim at computing isochrones that tell if a place is reachable on average

in a given time span. Since there is still the possibility that something is very fast reachable for only a relatively small amount of time within a time span, the term "time-invariant" isochrone is introduced and explained in detail.

Generalizing Dataset Creation. Both, the enhancements to isochrones and the improvements regarding their computation, can not be performed without datasets that describe networks the reachability analysis is performed in. Since there are no standards that describe how to build Multimodal Spatial Networks, a way to gather information from various sources and to join it into one appropriate model is introduced. The data sources have to at least deliver information about the road network, elevation information and the schedules from public transportation systems. To minimize the effort to build such datasets they are provided together with the creating software application as open-source.

User Interface Implementation. One of the biggest advantage of the world wide web is the possibility to make information available for everybody. Therefore, it is sensible to provide a visual interface that allows for displaying calculated isochrones. The visualization of the reachable area within an isochrone and simple analysis based on it can be done using the proposed web application. The deployment also allows to check evaluated algorithms and their computation results at least on a basic level.

1.4. Publications

Throughout the PhD studies several research papers and posters haven been published at national and international scientific conferences. In particular, the following work has been published:

Conference Papers

- N. Krismer, D. Silbernagl, G. Specht and J. Gamper. Computing Isochrones in Multimodal Spatial Networks using Tile Regions. In Proceedings of the 29th International Conference on Scientific and Statistical Database Management (SSDBM), pages 33:1–33:6, June 2017, Chicago, Illinois, USA [65]

- N. Krismer, D. Silbernagl, J. Gamper and G. Specht. osmPti2mmds – Erstellung von multimodalen Datensets aus OpenStreetMap und ÖPNV-

Informationen. AGIT Journal, pages 185–190, July 2016, Salzburg, Austria [64]

Conference Posters

- N. Krismer, J. Gamper and G. Specht. Reachability calculation based on average isochrones regarding time distribution. AGIT, July 2015, Salzburg, Austria [63]

- N. Krismer, G. Specht and J. Gamper. Isochrones in Multimodal Spatial Networks. AGIT, July 2014, Salzburg, Austria [62]

Workshop Contributions

- N. Krismer, D. Silbernagl, M. Malfertheiner and G. Specht. Elevation Enabled Bicycle Router Supporting User-Profiles. In Proceedings of the 28th GI-Workshop Grundlagen von Datenbanken (GvDB), pages 74–79, May 2016, Nörten Hardenberg, Germany [66]

- N. Krismer, G. Specht and J. Gamper. Incremental Calculation of Isochrones Regarding Duration. In Proceedings of the 26th GI-Workshop Grundlagen von Datenbanken (GvDB), pages 41–46, October 2014, Bozen-Bolzano, Italy [67]

Other Contributions

- N. Krismer. Interaktive Karten und HTTP/2. Freie und Open Source Software für Geoinformationssysteme (FOSSGIS), July 2016, Salzburg, Austria [58]

- N. Krismer. INNsochrone. 4. Tiroler IT-Day, May 2015, Innsbruck, Austria [57]

Co-authored publications

- D. Silbernagl, N.Krismer, N. Augsten and G. Specht. Recommending OSM Tags To Improve Metadata Quality. In LocalRec'17:1st ACM SIGSPATIAL Workshop on Recommendations for Location- based Services and Social Networks, 10 pages, November 2017, Los Angeles, California, USA [113]

- D. Silbernagl, N. Krismer and G. Specht. Comparing OSM Area-Boundary Data to DBpedia. In Proceedings of the 12th International Symposium on Open Collaboration, pages 11:1–11:4, OpenSym 2016, August 2016, Berlin, Germany [115]

- D. Silbernagl, N. Krismer and G. Specht. osmpg2java – Konvertierung von OSM-Datenbankelementen zu JTS-Objekten. AGIT Journal, pages 179–184, July 2016; Salzburg, Austria [116]

- D. Silbernagl, N. Krismer, M. Malfertheiner and G. Specht. Optimization of Digital Elevation Models for Routing. In Proceedings of the 28th GI-Workshop Grundlagen von Datenbanken (GvDB), pages 103–108, May 2016, Nörten Hardenberg, Germany [114]

1.5. Thesis Outline

The remainder of this thesis is structured as follows. The subsequent Chapter 2 describes the related work that is ongoing in the field of reachability analyses using isochrones. It distinguishes between scientific contributions as well as non-scientific. The later ones are further divided into commercially invented and non-commercially developed work. Chapter 3 gives some definitions and shows some basics used in the context of Multimodal Spatial Networks and isochrone computation.

Chapter 4 outlines algorithms capable to create isochrones in Multimodal Spatial Networks. Chapter 5 highlights innovations made regarding algorithms during the PhD studies. Chapter 6 addresses the second research objective and lists enhancements made to the application capabilities of isochrones. These include usage of elevation data and tailoring results to individuals. Chapter 7 gives an overview of the workflow that is used to create datasets representing Multimodal Spatial Networks. The web application developed throughout the PhD studies is described in Chapter 8.

To examine the outcomes of the algorithms discussed in Chapter 4 and 5 as well as for the enhancements made in Chapter 6 empirical evaluations are listed in detail in Chapter 9. The thesis closes with Chapter 10 that concludes the entire thesis and gives possible further research directives.

CHAPTER 2

Related Work

In this chapter an overview of work done in the field of isochrone computation is given. Various contributions are listed ranging from scientific publications dealing with algorithms and result visualization to non-scientific ones. Distinguishing between open-source and non-free products unimodal as well as multimodal isochrones are discussed. Networks allowing for multimodal movement are further examined for the fact if public transit schedules are used. The systems are analyzed if they are able to make correct estimations even on weekends and holidays or if aggregated information is used. Furthermore, visualization of possible results is shortly discussed.

2.1. Publications

The basis of every isochrone and isodistance computation are algorithms that were originally developed for the application of routing. However, not all algorithms from this domain are suitable for isochrone calculation, since contrary to navigation or routing only a starting or a destination point is known, but not both locations. This excludes some algorithms optimized for routing, like the A* algorithm [46], to be used in Multimodal Spatial Networks. Therefore,

when it comes to exact isochrone creation in such networks, the results are typically computed in a Dijkstra like fashion.

The algorithm "MDijkstra" (*M*ultimodal *Dijkstra*) developed by Bauer et al. [9] is an adapted version of the one presented by Dijkstra in 1959 [23] and was modified to work with multimodal information. It loads the entire spatial network from the underlying data source into memory and performs the computation solely in memory. Based on this approach Gamper et al. developed "Mine" (*M*ultimodal *i*ncremental *n*etwork *e*xpansion) in 2011 at the Free University of Bozen-Bolzano (FUB) [33]. To achieve the goal of using less memory data is loaded into memory only on demand. One year later the same authors reduced the memory footprint by pruning processed information from memory after it was used for computation. The resulting strategy is termed "vertex expiration" and was first used in "MineX" (*M*ultimodal *i*ncremental *n*etwork *e*xpansion using vertex e*X*piration). This algorithm has been proven to be optimal in terms of memory management [34]. The drawback of the latter two approaches is the need for a fast data source connection, since the number of requests is much higher when compared to the base approach "MDijkstra". These algorithms are discussed in more detail throughout Chapter 4 and will also be included in the evaluations in Chapter 9.

Publications about applications that allow for easy isochrone determination have also been published by M. Innerebner in 2013. The paper "ISOGA: a System for Geographical Reachability Analysis" describes the usage of a web browser to create isochrones within specific cities and to perform geospatial analyses with the result [52].

There have been more ideas by the same research group from Bolzano that were recorded in the PhD thesis of M. Innerebner which includes a description about an algorithm using spatial range queries [51]. Vertices are loaded using circular ranges from the underlying database. These ranges shrink as remaining travel time decreases and as a result propose a hybrid approach between using minimal memory (Mine/MineX) and minimizing request to the data source (MDijkstra).

Several publications using isochrones as basis for spatial analyses have been published by other researches as well. Those include "nawo" [120, 123] or "City Focus" [3, 37] that allow for finding real estates by filtering for certain criteria like the time (or distance) to the next school, park or to public transport facilities. These type of questions are called "optimal location queries" (OL queries) [16, 25, 131]. One way to answer them, is to intersect multiple isochrones.

Approaches based on recent algorithms, like Contraction Hierarchies (CH) or Custom Route Planning (CRP), rely on network splitting. Since the splitting process becomes very complicated when using multimodal networks and especially hard when schedules are involved, they only have been applied to unimodal isochrone computations [10] and computation is done solely in-memory. Unimodal isochrones are also computed at the University of Heidelberg, where A. Zipf and P. Neis developed the application "OpenRouteService" [85, 87]. Though originally planned for fast route finding by using Volunteered Geographic Information (VGI) [39] from OpenStreetMap the web application enables isodistance and unimodal isochrone computation since early 2017.

Regarding the visualization of isochrones, the main task is to compute a border around all the vertices and edges in the network that are part of the result, referred to as isoline. This is similar to the problem of finding an outline to a set of points. Several approaches deal with this problem, including two algorithms that have been implemented at the Free University of Bolzano, both published in 2010 by S. Marciuska [73]. Other solutions include convex [1, 26, 41, 53] and concave hull [81] computation, alpha-shapes [28, 29], algorithms based on Voronoi diagrams and Delaunay triangulations [7, 77, 132] and others [93]. The fact that isochrones in Multimodal Spatial Networks possibly cover multiple point sets (disconnected parts of the underlying graph), is not addressed by all the algorithms. Therefore, a preprocessing step to determine the different point sets has to be included, that then allows to use the algorithms on the separated parts of the graph.

2.2. Systems

Taking academic publications aside, isochrones and isodistances can be computed using common software and services. This section lists open-source and commercially available approaches. It starts from the ones being able to compute unimodal isochrones and continues with those creating multimodal isochrones. At last, systems are presented that work in Multimodal Spatial Networks and regard schedules of the public transportation system.

A solution for unimodal isochrones is implemented in the so called "pgRouting" software [103]. It builds on the spatial database extension "PostGIS" [104] that itself is an extension to the open-source database "PostgreSQL" [119]. The computation of so called "driving distances" (meaning the same as isochrones) is implemented using the Dijkstra algorithm. The results from pgRouting can be visualized using geographical information system (GIS) desktop clients like

QGIS [42]. Isodistances can be computed solely by GIS desktop clients, e.g. with GRASS and its v.net.iso component [88].

Though originally developed for unimodal isochrone computation, pgRouting can be used to compute multimodal isochrones by reorganizing the underlying spatial data and merging all modalities into a single graph. This has been shown for the city of Vienna by Anita Graser some years ago in 2013 [43]. The disadvantage of solutions based on pgRouting is the lack of knowledge about the schedules of the public transportation system. Information on how to get from one station to another is not based on actual date or time, but on travel duration between stations. This makes it impossible to regard changing schedules on weekends, holidays or for cities where the frequency of a service depends on the daytime.

When the multimodal nature of a dataset needs to be addressed, but the exactness of the results is not of great importance, services from Rome2rio [108], Mapumental (which is also known as "mySociety travel-time maps") [83] and the "DB Umkreissuche" of the Deutsche Bahn [21] are available. Rome2rio also shows the service frequency as part of the results, but still fails to vary it between days (e.g. between a working day and a holiday) or certain times of the day. Mapumental and the DB Umkreissuche are very limited when it comes to geographical coverage. Right now, Mapumental is only available for the home country of the mySociety initiative, namely Great Britain. The DB Umkreissuche, which itself is based on the "HACON Umkreissuche" [44], only works in the region of Germany.

Mapnificent [126] is an open-source approach very similar to the services mentioned before. In contrast, it uses data from OpenStreetMap and schedules provided in the General Transit Feed Specification (GTFS) format [2, 75].

A commercial approach to allow for multimodal routing is implemented by the "Verkehrsauskunft Österreich (VAO)" [124] for the region of Austria. Although this seems to include some clever multimodal approaches (e.g. so called "Park-and-Ride" solutions where it is only possible to drive by car to a train station but not traveling by car after leaving the train), this services allow for plain routing, but not for isochrone determination. Although information about the road network that is used by the VAO (the Graphintegrations-Plattform GIP [99]) has been released to the public in 2016, the underlying algorithms and also the data about the public transportation system are not available. As a result, extending the algorithms is not possible. Multiple applications that are provided by companies and enable routing in Austria, are based on the GIP and VAO (e.g. web applications "Routenplaner IVB",

"Fahrplan VVT", "ÖAMTC Verkehrsauskunft" or "Routenplaner bmvit") [124].

An open-source project aiming at multimodal routing is OpenTripPlanner (OTP) [97]. It includes a component called "OTP Analyst" that is able to compute multimodal isochrones. Additionally, it does take the schedules of the transportation system into account (delivered as GTFS file) and therefore is able to deliver different isochrones depending on the starting date or time. However, OTP uses a precomputed graph structure created from OpenStreetMap data from which isochrones can be extracted. This makes its memory footprint quite large and unsuitable for large networks. For the city of New York the web application "WNYC Transit Time NYC" [130] has been implemented using OpenTripPlanner. To keep the network small, the region has been reduced to 2930 hexagons between which multimodal travel times are computed.

Recent developments by the people behind OTP include Conveyal Analyst [17]. It implements the same features as OTP Analyst, realized with more modern technologies. In addition, it adds the possibility of so called scenarios, allowing decision makers to simulate multiple "What-If" scenarios. This can be of great benefit when planning maintenance or when placing stations to build. The source of both tools can be found at github [18, 98]. In contrast to OTP, which is licensed under the GNU Lesser General Public License (LGPL), Conveyal Analyst is published under the Apache 2 license.

Another open-source project, named "GeoTrellis Transit" [4] is based on GeoTrellis by Azavea. It uses Apache Spark and Scala to perform analyses on raster data. Multimodal isochrones are one of them, with the uniqueness of using accurate times, but categorizing on days of a week. Since no exact date can be entered, one has to choose between "weekday", "saturday" or "sunday", making it impossible to regard holidays. Since a raster builds the basis of all the analyses the results are not as accurate as they would be when using vector data.

Similar to OTP there is another solution that started from routing and enables isochrone computation. Graphhopper is a software library also licensed under the Apache 2 license and enables easy integration into other projects [109]. However, the isochrone API which is built on top of the very same library, is closed source and only available against a fee. Although in July 2017 it was announced that it will become open-source and freely available, the code has not been released by the end of October 2017.

Other solutions capable to compute multimodal isochrones with regard to a certain start date and/or time are known to be available from Route360°/Motion Intelligence GmbH, a company located in Berlin, Germany [82]. Customer solutions relying on their services, like Naturtrip [84], seem to be very fast and use state of the art techniques. As can be seen from some published code experiments with HTML5, WebGL and Vector Tiles are used. Since isochrone computation is available as part of an API only and no code about it has been published, nothing is known about the underlying algorithms or techniques used to compute the results.

Direct competitors to Route360° include Walk Score's Professional Travel Time [125], the TravelTime Platform by iGeolise [49] and iso4app [68]. Besides some small projects that work with Java and Docker available on github [50], sources from iGeolise are not available. For Walk Score and iso4app only API documentations and client libraries are available, but nothing is publicly available on the algorithms used. The consumers are able to compute isochrones solely by web service.

The visualization of the result in the non-scientific field ranges from basic techniques that draw a circle with a radius defined by the remaining time around public transportation stations (Mapnificent) to advanced algorithms like Duckham et al. (OpenTripPlanner). Newer approaches often utilize the GPU of computer systems (Route360°), sometimes reducing the needed effort by rastering results (GeoTrellis Transit). What can be noticed is that all solutions do not generate exact results. While this might be clear for the latter approach or when drawing simple circles around stations, it has to be noted, that even convex/concave hull computation or results created with the help of Voronoi or Delaunay triangulations must not be exact (for Duckham et al. the precision depends on the so called length parameter).

One of the more surprising facts is that companies heavily involved in map generation do not provide good solutions to compute isochrones. Google for example does not provide anything regarding isochrones in Google Maps (neither in APIs nor on the website itself). Approaches using technologies from Google are heavily based on picking random points and computing travel time distances to those. This is done until the travel duration reaches the appropriate travel time and repeated for multiple cardinal directions. If enough directions have been computed the resulting points are connected. The result is then claimed to be a simulated isochrone [101], but is by far less accurate and therefore not comparable to the methods and services described before. A similar approach has also been implemented with Bing Maps [13]. In fact this can be done with every map provider that allows routing through an API. However, this approach will not be incorporated, since it imposes multiple

problems. It is not only less accurate, it also does not deal with multimodality correctly. Even if the underlying routing itself is carried out in a multimodal way, results will always be connected. This is not correct, since multimodality typically leads to reachability islands at stations of the public transportation system.

CHAPTER 3

Defining Isochrones

This chapter covers the basic definitions on which the rest of this thesis is build on. First, a formalization of the term "Multimodal Spatial Network" (MMSN) and all the terms it builds on are given. It is further distinguished between a reachable area and an isochrone. After listing samples, definitions in the field of geographical networks are given. This is done in order to allow the computation of isochrones also within real-world datasets rather than solely relying on mathematically modeled ones.

3.1. Networks and Graphs

In our modern world networks are everywhere. A well-known example for a network is the internet. It is the technical network between multiple computers and consists of the words "Inter", which is Latin for between, and the abbreviation "net" for network. There are various other networks like the social networks or even the neuronal network of the human brain. The noun "network", as defined within Googles knowledge graph (which is a network on its own), is "a group or system of interconnected people or things" [40]. There are several types of different networks, like biological networks, technical and

computer networks, social networks, advertising networks and many more. One definition for the theory about networks was given by John Baez, mathematical physicist at the University of California (U.C Riverside) as "the study of complex interacting systems that can be represented as graphs equipped with extra structure" [5]. Although the word "graph" itself was first used by J.J. Sylvester in 1878, graph theory goes back to Leonard Euler and the 18th century [12]. A graph has been defined in mathematics as a set of vertices (sometimes also called nodes) that are connected by edges (sometimes also referred to as arcs or lines). Trudeau defined it as "a graph consists of two sets named the vertex set and the edge set. The vertex set is a nonempty set, and the edge set may be empty, but if not it contains two-element subsets of the vertex set" [122]. A graph can be defined as follows.

Definition 1: Graph

A *graph* $G = (V, E)$ is a pair of sets V and E. The set V is non-empty and called the vertex set (or node set), while set E is the edge set of graph G. An edge from E is a two-element subset of vertices from V.

There are also mathematical representations used that define a graph as a triple. In these definitions an additional function, the so called incidence function, is added that associates with each edge of G an unordered pair of (not necessarily distinct) vertices of G. This definition is used for example by Bondy [14] or West [127]. Since both definitions are used throughout literature, Definition 1 will be used throughout this thesis. An example for a graph can be found in Figure 3.1. The vertices in this case are named *v1*, *v2* and *v3*.

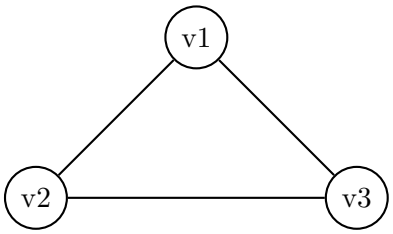


Figure 3.1.: Sample graph.

Graphs can be further categorized by various criteria. Some, but non all, of these categorizations are listed in the following lines. First, a graph can be distinguished by the fact that its edges can either be directed or not. Edges are typically named by the vertices they connect. Therefore, edges are referred to as *v1v2*, *v2v3* and *v3v1*. If edges do not have an orientation (which means that $vw = wv$) the graph is called an undirected graph.

Definition 2: Undirected graph

If the edges in a graph G are unordered pairs of vertices, the graph is called a *undirected graph*.

Figure 3.1 shows an undirected graph, since edges without specific direction have been used. If edges are oriented and an edge from vertex v to vertex w is not the same as the one from vertex w to v (written as $vw \neq wv$) the graph is called directed (or digraph).

Definition 3: Directed graph

If the edges in a graph G are ordered pairs of vertices, the graph is called a *directed graph*.

An example for a directed graph is shown in the following Figure 3.2. For an edge $(v1, v2)$ in a directed graph, the vertex $v1$ is called the start vertex and $v2$ denotes its end vertex.

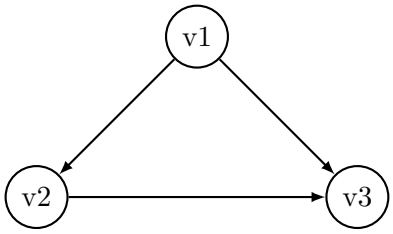


Figure 3.2.: Directed graph.

Graphs also allow for two vertices u and v to be connected by more than two edges. This means that in every direction there can be zero to arbitrary many edges, including edges that start and end at the same vertex. These edges are called multiple edges (or parallel edges).

Definition 4: Multiple edges

Edges connecting the same pair of vertices are called *multiple edges*.

Directed graphs can easily be created from undirected graphs with no semantic change. All that needs to be done is to replace each undirected edge with two directed edges modeling opposite directions resulting in multiple edges. Applying this method to the graph from Figure 3.1, the result looks like the following Figure 3.3. The edge count has doubled, so that the edges now are *v1v2*, *v2v1*, *v2v3*, *v3v2*, *v3v1* and *v1v3*.

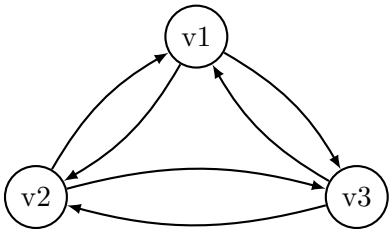


Figure 3.3.: Directed graph containing multiple edges.

The edges in a graph are allowed to start and end at the same vertex. If this is the case the edge is called a loop. The following Definition 5 has been used by [127].

Definition 5: Loop

A *loop* is an edge whose endpoints are equal.

In Figure 3.4 a directed graph is shown that contains multiple edges and loops. The loop in the graph connects $v2$ with itself. In this example also multiple edges have been used, that have the same direction (edge $v1v3$).

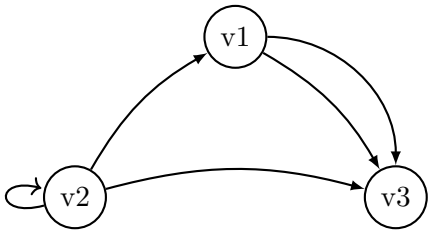


Figure 3.4.: Directed graph containing multiple edges and loops.

A loop is not be confused with a cycle. In contrast to a loop, a cycle is a closed walk in a graph starting and ending at the same vertex, i. e $v1, \ldots, vk, v1$ with $v1, \ldots, vk$ all distinct, and $k \geq 3$. A **"simple graph"** does neither include loops nor multiple edges. If a graph G is allowed to contain multiple edges and loops the graph is called a **"general graph"**. So every simple graph is a general graph, but not every general graph is a simple graph [128]. To be more precise, a graph containing multiple edges, but no loops is called a **multigraph**. If multiple edges and loops are allowed, then the graph is called a **pseudograph** [6]. Furthermore, if edges are directed then the graphs are also referred to as a **multidigraph** or **pseudodigraph** depending on the inclusion of loops.

Furthermore, a graph can be called "weighted" if there is a number associated to each edge. The number is called the weight w of the edge e. A possible formalization of weighted graphs has been given by J. A. Bondy [14].

Definition 6: Weighted graph

With each edge e of G let there be associated a real number $w(e)$, called its weight. Then G, together with these weights on its edges, is called a *weighted graph*.

Weights can have different meanings in a graph. The can be used as a traversal cost or as an indicator how important the edge is (e.g. in a friendship graph the weight could indicate how strong a friendship is) or even some activation potential (meaning that the edge is only valid if a certain precondition can be met). An example of a weighted graph is given in Figure 3.5.

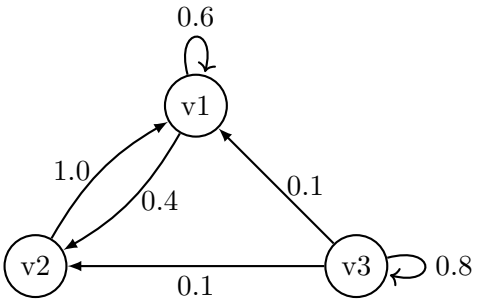


Figure 3.5.: Weighted multidigraph.

A generalization for graphs is called hypergraph. It allows edges to connect more than two vertices. Such edges are called hyperedges. These terms have been defined for example by Papa and Markov [102].

Definition 7: Hypergraph

A *hypergraph* is a pair of sets $H = (V, E)$ where V is the set of vertices of the hypergraph and E is the set of hyperedges of the hypergraph. Each hyperedge is a non-empty subset of V, the size of this subset is called the hyperedge's degree.

For road networks the hyperedges need to be ordered. Since hypergraphs allow a hyperedge to be a subsets of V, the definition is not strict enough. The appropriate graph to use is a oriented k-uniform hypergraph, as defined by Frankl [31]. The definition is basically the same as for general hypergraphs with the difference that each hyperedge is a k-tuple of distinct vertices (instead

of a non-empty subset of V). They are often also called hypertournaments or oriented hypergraphs.

An example of a hypertournament containing multiple edges and loops is shown in Figure 3.6. In this graph the hyperedge starting at $v3$ and ending in $v2$ has degree four and connects the vertices $v3$, $v22$, $v21$ and $v2$. The second hyperedge also starts at $v3$ and ranges to $v1$ via $v11$. Its degree is three.

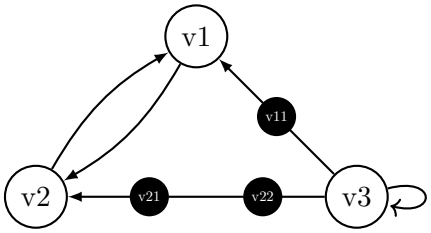


Figure 3.6.: Hypertournament with loops and multiple edges.

For isochrone computation the most important terms of graph theory have nearly all been covered. There are only some minor additional properties, that will now be covered very shortly.

The degree of a vertex is given by Bondy [14] while additions about in- and out-degree have been made by V. K. Balakrishnan [6] and by T. H. Cormen [19].

Definition 8: Degree

The *degree* $d_G(v)$ of a vertex v in G is the number of edges of G connected with v, each loop counting as two edges. In a directed graph, one may distinguish the in-degree (number of incoming edges) and out-degree (number of outgoing edges).

The size and order of a graph are a first indicator when it comes to graph comparison. Definitions for those properties are taken from [6].

Definition 9: Order & Size

The *order* of a graph G is the number of its vertices. The *size* of a graph G is the number of its edges.

3.2. Spatial Networks

A spatial network is one where vertices and edges represent spatial elements associated with geometric objects. This is sometimes also called a geometric graph. There are multiple samples for spatial networks:

- Biological networks of neurons in the human brain or the structure of a molecule as well as representations of metabolic pathways [47]
- Galactic networks modeling galactic clusters, galaxies and stars
- River networks modeling the flow of rivers
- Transportation networks including road networks, railway networks and the subway networks

Isochrone computation can be performed in every spatial network, but since this thesis focuses on calculation of Multimodal Spatial Networks, the remainder targets on various transportation networks allowing multiple ways of transport. In such structures edges refer to street segments, transportation routes, escalators, walkways or railroads that connect vertices representing junctions, public places, bus stations and other points of interest. The vertices in the underlying graph are represented by a geographical position.

3.2.1. Transportation Networks

When it comes to transportation networks extra attributes connected to the graph are of interest and represent geographical location, road surface or the maximum speed allowed. Therefore, it makes more sense to write about a transportation or road network and not a street graph, although it is quite common to mix these terms. Furthermore, streets typically have an orientation (e.g. one-way streets), which means that a directed graph needs to be used to represent a road network. In a road network multiple edges are used for various purposes. If two junctions are connected by streets as well as a sidewalk this can be expressed with the help of two separate edges. Different means of transportation are modeled with the help of multiple edges, where one edge is used per transport mode. Loops can be used in road networks [90], although this depends on the exact implementation. Roundabouts at the end of a street could in theory be expressed by them, or the changing between different means of transportation at a train station could be realized with the help of loops, if the whole station is modeled as a single vertex. Changing from platform

1 to platform X could then be realized with this method. Even if there are different transportation systems at the same station available, a loop could be used to change between these systems. An example for this could be a station where trains and subways meet. If such a station is modeled using the very same vertex, a loop with a certain weight can model the time needed to change between different platforms or transportation systems.

Streets in a road network have a certain length which can be used to define the cost to traverse the edge. Also the maximum speed allowed (or both attributes) can be used to define such a "weight", which makes the usage of a weighted graph obligatory. To sum up, a road network is a spatial network that can be represented by a weighted multidigraph. A road network is not planar as often stated. In fact it is an so called almost planar graph, meaning that there are nearly no edges crossing each other when the underlying graph is drawn in the plain. Only the edges representing bridges and tunnels will cross each other, but compared to the size of the graph the number of crossing edges will be very small [110]. An example of a road network including sidewalks, footways, bicycle paths and streets is plotted in Figure 3.7.

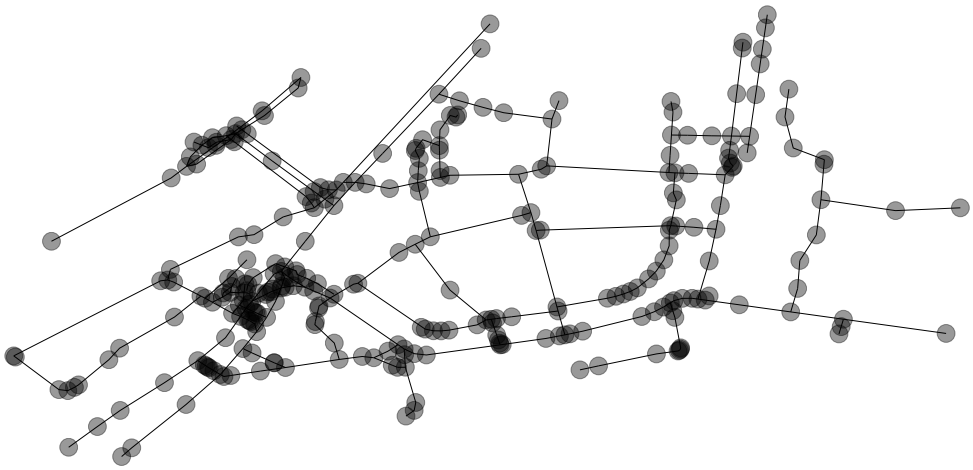


Figure 3.7.: Spatial road network modeling the city center of Innsbruck, Austria.

It can be seen that a graph is used to model this road network. The shown data is modeled using a planar one, but as soon as the river Inn or the walkable shores and beaches of it are taken into account, an almost planar graph needs to be used. As will be shown later in Chapter 7.1 the transportation network can best be modeled with an almost planar weighted hypertournament. The weights on the hyperedges equal the maximum allowed traveling speed. When it comes to routing and reachability computations, vertices are only of interest, if they represent junctions, correspond to change in weight or transport mode, or are connected to the public transportation system, i.e.

have at least one discrete space and time egde. By using a hypertournament computation performance can be increased, since only start and end vertices need to be considered, without losing information about the detailed shape of an edge (including the length of an edge).

3.2.2. Multimodal Spatial Networks

A Multimodal Spatial Network allows to model several transport systems, possibly representing different modalities, in a single network. In order to achieve this, each edge has to be connected to a transportation system, while multiple edges within the underlying graph can be used for connections of multiple transportation systems between the same vertices. This kind of network has been defined by Gamper et al. [33] and by M. Innerebner [51]. The definitions in this section are therefore based on and closely related to the ones given in these publications.

As already mention in Chapter 1 a transportation network can be classified as continuous and discrete in space and time, respectively. This leads to the following four different transport modes:

- *continuous space and continuous time mode* ($\mu =$ "*csct*"), e.g. pedestrian network
- *discrete space and discrete time mode* ($\mu =$ "*dsdt*"), e.g. the public transport system with its trains and buses
- *discrete space and continuous time mode* ($\mu =$ "*dsct*"), e.g. escalators or moving walkways
- *continuous space and discrete time mode* ($\mu =$ "*csdt*"), e.g. regions or streets that can be passed by pedestrians or cars only in specific time slots.

Definition 10: Transport mode

The *transport mode* function $\mu : \Theta \mapsto \{$"*csct*", "*csdt*", "*dsct*", "*dsdt*"$\}$ assigns to each transportation system a transport mode and the function $\theta : E \mapsto \Theta$ connects an edge with a transportation system.

To each transport mode using discrete time (either $\mu =$ "*dsdt*" or $\mu =$ "*csdt*") a schedule is applied that tracks the arrival and departure time at stations for the individual trips of a certain route. A *route* consists of multiple trips which are possibly spread over different times throughout the day. This is a grouping criteria for trips that in public transportation systems is also often

referred to as "Line". Consider a sample bus route ($\Theta = B$) named "R1" including stops at vertices v_2, v_3, v_6 and v_7 (and maybe more). A schedule for this transportation system and a simple Multimodal Spatial Network could look like the one shown in Figure 3.8 and Table 3.1.

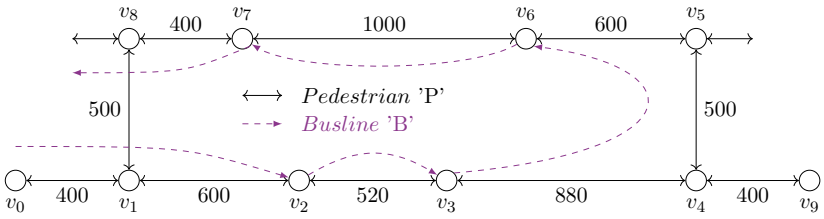


Figure 3.8.: Spatial Network containing two modalities.

T-Sys (Θ)	Route (R)	Trip (TID)	Stop (V)	Arrival (σ_a)	Departure (σ_d)
⋮	⋮	⋮	⋮	⋮	⋮
B	R1	1	v_2	05:30:00	05:30:00
B	R1	1	v_3	05:31:00	05:31:30
B	R1	1	v_6	05:33:00	05:33:00
B	R1	1	v_7	05:34:00	05:34:30
⋮	⋮	⋮	⋮	⋮	⋮
B	R1	2	v_2	06:00:00	06:00:00
B	R1	2	v_3	06:01:00	06:01:30
B	R1	2	v_6	06:03:00	06:03:00
B	R1	2	v_7	06:04:00	06:04:30
⋮	⋮	⋮	⋮	⋮	⋮

Table 3.1.: Example of a Schedule.

For computation purposes grouping trips together in routes is not of interest. Therefore, the route information can be omitted so that schedules are characterized by five attributes. A possible formalization of a schedule is therefore listed in Definition 11.

Definition 11: Schedule

$S = (\Theta, TID, BV, \sigma_a, \sigma_d)$ is called a *schedule* for the transportation system S, where TID is a set of trip identifiers, $BV \subseteq V$, and $\sigma_a : \Theta \times TID \times BV \mapsto \mathbb{T}$ and $\sigma_d : \Theta \times TID \times BV \mapsto \mathbb{T}$ determine arrival and departure time within the time domain $\mathbb{T}$.

A schedule follows a time-dependent model [8, 22, 106], where the cost of an edge is not just a scalar value, but a piece-wise linear function τ that maps each possible arrival time from the start vertex of the edge to a travel cost.

The function τ is later formalized in Definition 12. For an edge $e = (u, v)$ this function computes the time-dependent transfer time that is required to traverse e, when starting from u as late as possible and arriving at v no later than time t. Pyrga et al. [106] term this the *latest-departure problem* (LDB), in which the optimization criterion is to maximize the actual departure time at the departure station among all connections that arrive at the arrival station by the given arrival time. This problem occurs when an isochrone is computed as the set of points from where the query point is reachable within the given time constraints. The opposite problem is termed *earliest-arrival problem* (EAP) [106], in which the optimization consists in minimizing the difference between the arrival time and the given departure time. This problem occurs when isochrones are defined as the set of all space points that are reachable from the query point.

The information from a schedule is thus used to compute the time needed to traverse an edge. For discrete time edges (i.e., $\mu(e) =$ "*dsdt*") the arrival or departure time must be considered. If the departure time is given, the transfer time is the difference between the earliest arrival time t' at v minus t. If the arrival time at v is given, the transfer time is the difference between the current time t and the latest departure time t' at vertex u. For continuous time edges (i.e., $\mu(e) =$ "*csct*"), the transfer time is modeled by a traveling speed $s : E \mapsto \mathbb{R}^+$, an edge length $\lambda : E \mapsto \mathbb{R}^+$ and a time-dependent weight function $\omega : E \times \mathbb{T} \mapsto \{(0, 1], \infty\}$. The weight is a function that assigns in dependency of the time to each edge a value from the interval $(0, 1]$ or ∞ (if the edge cannot be traversed). The time needed to traverse an edge e is called transfer time τ, which is defined in Definition 12.

Definition 12: Transfer time

Let $t \in \mathbb{T}$ be either the arrival time at v or the departure time at u. The *transfer time* of edge $e = (u, v)$ for the transport modes "*dsdt*" and "*csct*" is determined by a function $\tau : E \times \mathbb{T} \to \mathbb{R}^+$ that is defined as follows:

$$\tau((u,v),t) = \begin{cases} \frac{\lambda(e)}{s}\omega(t) & \mu(\theta(e)) = \text{"}csct\text{"} \\ t - t' & \mu(\theta(e)) = \text{"}dsdt\text{"} \wedge t \text{ is arrival time at } v \wedge \\ & t' = \max\{\sigma_d(r,tid,u) \mid \sigma_a(r,tid,v) \leq t\} \\ t' - t & \mu(\theta(e)) = \text{"}dsdt\text{"} \wedge t \text{ is departure time at } u \wedge \\ & t' = \min\{\sigma_a(r,tid,u) \mid \sigma_d(r,tid,v) \geq t\} \end{cases}$$

For the transport modes "*csdt*" and "*dsct*" the transfer time can be specified in a similar way.

Assuming a constant walking speed of $s = 4\,\mathrm{m/s}$ and a constant weight $\omega(t) = 1$, which yields a transfer time $\tau(e,t) = \frac{\lambda(e)}{4\,\mathrm{m/s}}$ for pedestrian edges. In Figure 3.8 with a given starting time 06:02:00, the transfer time on the pedestrian edge (v_6, v_7) is $\frac{1000\mathrm{m}}{4\,\mathrm{m/s}} = 250\,\mathrm{s}$. The transfer time on the bus edge (v_6, v_7) is 06:04:00 − 06:02:00 = 120 s, including a 60 s waiting time for the next bus.

With the definitions of a transport mode, schedules and the transfer time given in Definition 10, 11 and 12, it is now possible to define a Multimodal Spatial Network (MMSN) that is able to model a transportation network allowing multiple ways of transport.

Definition 13: Multimodal Spatial Network

A *Multimodal Spatial Network* is an eight-tuple $\mathsf{N} = (G, \Theta, S, \theta, \mu, \lambda, \tau, \omega)$, with

- $G = (V, E)$ is a directed graph representing a routable spatial network.
- Θ is a set of transport systems.
- S is a schedule as defined in Definition 11.
- The function $\theta : E \mapsto \Theta$ connects a transportation system to each edge
- μ assigns a transport mode to each edge as formalized in Definition 10.
- $\lambda : E \mapsto \mathbb{R}^+$ records an edge's length
- τ: the transfer time as defined in Definition 12.
- $\omega : E \times \mathbb{T} \mapsto \{(0, 1], \infty\}$ assigns a weight to an edge.

The simple network shown in Figure 3.8 can be seen as a representative of a Multimodal Spatial Network.

3.2.3. Calculations in Multimodal Spatial Networks

Isochrone calculations in a Multimodal Spatial Network can generally be done in two directions. Either starting from a certain query point reaching out into the network, which is termed "outgoing" computation, or ending at a query point, termed "incoming" isochrone calculation. This corresponds to the differentiation of the latest-departure and the earliest-arrival problem described by Pyrga et al. [106] which has to be taken into account for network edges with discrete time transportation systems (μ = "*dsdt*" and μ = "*csdt*"). For

the remainder of this subsection only outgoing isochrones will be considered. Incoming computations are done in a similar way.

At first, the query point of a computation needs to be defined. It potentially can be any point, so it is not guaranteed to coincide with a vertex or lays on an edge of the spatial network.

Definition 14: Query point

A *query point* q is any point represented by coordinates.

A location, in contrast to a query point, is used to represent accessible parts of the network. It is defined in Definition 15.

Definition 15: Location

A *location* in a Multimodal Spatial Network N is any point on an edge $e = (u, v) \in E$ that is accessible. It is written as $l = (e, o)$, where $0 \leq o \leq \lambda(e)$ is an offset that determines the relative position of l from u.

A location coincides with vertex u if $o = 0$ and vertex v if $o = \lambda(e)$; any other offset refers to an intermediate point on edge e. In continuous space networks all points on the edges are accessible. Since a pedestrian segment is modeled as a pair of directed edges in the opposite direction, any point on it can be represented by two locations, $((u, v), o)$ and $((v, u), \lambda(u, v) - o)$, respectively. For instance, in Figure 3.9 the location of q is $l_q = ((v_2, v_3), 180) = ((v_3, v_2), 80)$. In discrete space networks only vertices are accessible, thus $o \in \{0, \lambda(e)\}$ and locations coincide with vertices.

The first step of a calculation within a Multimodal Spatial Network is to project query points to locations. There can be multiple different cases:

- The query point q coincides with a vertex u of the graph G if the offset $o = 0$. The location representing q therefore is $l = (e, 0)$.

- The query point does not coincide with a vertex, but is accessible from an edge of the graph g (and therefore can be represented easily by a location). No projection of q is needed, only the offset has to be determined. To do so, the length from the query point to the start- (o_{start}) and end vertex (o_{end}) of the edge is computed. The smaller offset is used to create the location, so that $l = (e, min(o_{start}, o_{end}))$.

- In any other case the query point q needs to be projected onto a location l. This projection can be done, by finding the continuous space edges in the graph G that are nearest to q. After checking the transport mode, the projection is done by using an orthogonal line from the edge through the query point. The crossing point of the orthogonal line and the edge itself is the location that q is projected on. The offset of the location l is determined as described above.

As can be seen from locations, in a spatial network, not only edges themselves, but also accessible locations on edges are of interest. The part of an edge between two locations on the very same edge is referred to as edge segment as defined in Definition 16.

Definition 16: Edge segment

An *edge segment*, (e, o_1, o_2), with $0 \leq o_1 \leq o_2 \leq \lambda(e)$ represents the contiguous set of space points between the two locations (e, o_1) and (e, o_2) on edge e. The length function for edge segments is generalized as $\lambda((e, o_1, o_2)) = o_2 - o_1$.

Computations in a transportation network, such as way findings or reachability analyses, typically deal with more than only one edge or edge segment. In most of the cases a routing result ranges over more edges. Such a union of edges and edge segments is called path. Its definition is given in Definition 17.

Definition 17: Path

A *path* from a source location $l_s = ((v_1, v_2), o_s)$ to a destination location $l_d = ((v_k, v_{k+1}), o_d)$ is defined as a sequence of connected edges and edge segments, $p(l_s, l_d) = \langle x_1, \ldots, x_k \rangle$, where $x_1 = ((v_1, v_2), o_s, \lambda((v_1, v_2)))$, $x_i = (v_i, v_{i+1})$ for $1 < i < k$, and $x_k = ((v_k, v_{k+1}), 0, o_d))$.

The first and the last element in a path can be edge segments, whereas all other elements are entire edges. Edges along a path may belong to different transport systems, which implies a switch into a different transport system.

In Figure 3.9, a path from q to v_7 is to walk to v_3 and then to take the bus 'B' from v_3 to v_7, i.e., $p(l_q, v_7) = \langle x_1, x_2, x_3 \rangle$, where $x_1 = ((v_3, v_2), 0, 80)$ is an edge segment and $x_2 = (v_6, v_3)$ as well as $x_3 = (v_7, v_6)$ are complete edges. The visual representation of this path is illustrated in Figure 3.9 as follows.

To compare paths, a criteria has to be defined that can be calculated for every path. Different measures exist that can be used, i.e. the path length, the

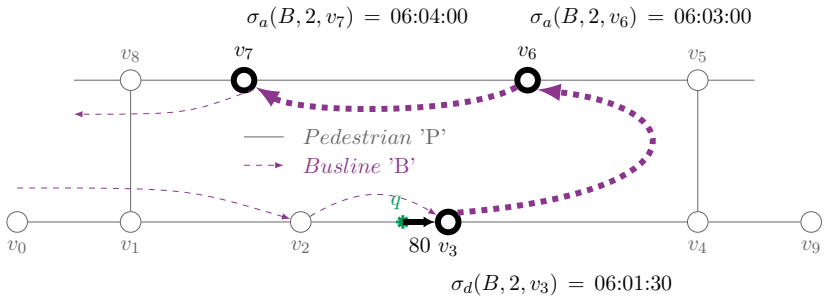


Figure 3.9.: Path in a network.

number of different modalities of the path or its average travel speed could be used. When it comes to isochrone computation, the needed time for traversal is the most important part, so the criteria builds on the transfer time of the edges included in the path. With an arrival time (or departure time depending on the computations direction) t at the location l_d the *path cost* is defined as follows in recursive Definition 18.

Definition 18: Path cost

The *cost* of a path $p(l_s, l_d) = \langle x_1, \ldots, x_k \rangle$ with t as arrival time at location l_d (or departure time at location l_s) is the sum of the individual transfer times τ of all edges and edge segments (x_1 to x_k) in the path. The number of edges is referred to as k, i.e.

$$\gamma(\langle x_1, \ldots, x_k \rangle, t) = \begin{cases} \tau(x_k, t) & k = 1 \\ \tau(x_k, t) + \gamma(\langle x_1, \ldots, x_{k-1} \rangle, t - \tau(x_k, t)) & k > 1 \wedge \\ & t = \text{arr. time at } l_d \\ \tau(x_1, t) + \gamma(\langle x_2, \ldots, x_k \rangle, t + \tau(x_1, t)) & k > 1 \wedge \\ & t = \text{dep. time at } l_s \end{cases}$$

If the path consists of a single edge or edge segment ($k = 1$), function τ computes the cost of traversing this edge, depending on whether t is arrival or departure time. If the path $\langle x_1, \ldots, x_k \rangle$, contains more than one edge or edge segments, the transfer time for the last edge (or edge segment), x_k, is determined if t is arrival time at l_d and the cost of the first edge (or edge segment), x_1, if t is departure time at l_s. For the remaining path, the function γ is called in a recursive way with a new arrival or departure time. The recursion terminates when the path only contains a single edge (or edge segment).

With the help of the Definition 18, the cost of the path shown above in Figure 3.9 can now be computed. The resulting cost is the one of the path $p(l_q, v_7) = \langle((v_2, v_3), 0, 160), (v_3, v_6), (v_6, v_7)\rangle$ using a departure time $t = 06{:}00{:}00$ at location l_q and a constant walking speed of $4\,m/s$.

The cost of traversing this path (where t' is the earliest possible departure time) is:

$$\begin{aligned}
\gamma(p(l_q, v_7), 06{:}00{:}00) &= \tau(((v_2, v_3), 0, 160), 06{:}00{:}00) + \gamma(p(v_3, v_7), t') \\
&= 40\,s + \gamma(\langle(v_3, v_6), (v_6, v_7)\rangle, 06{:}00{:}00 + 40\,s) \\
&= 40\,s + \gamma(\langle(v_3, v_6), (v_6, v_7)\rangle, 06{:}00{:}40) \\
&= 40\,s + \tau((v_3, v_6), 06{:}00{:}40) + \gamma(\langle(v_6, v_7)\rangle, t') \\
&= 40\,s + (50 + 90)\,s + \gamma(\langle(v_6, v_7)\rangle, t') \\
&= 40\,s + 140\,s + \gamma(\langle(v_6, v_7)\rangle, 06{:}03{:}00) \\
&= 40\,s + 140\,s + 60\,s \\
&= 240\,s
\end{aligned}$$

In line 5 the calculation of the transfer time on the discrete edge (v_3, v_6) with departure time at v_3 uses the earliest departure after 06:00:40. Since the first bus that matches this constraint departs at 06:01:30 and arrives at 06:03:00, we have a waiting time of 50 seconds at v_3, and the transfer time is $\tau((v_3, v_6), 06{:}00{:}40) = 50\,s + (06{:}01{:}30 - 06{:}03{:}00) = 140\,\text{s}$. Thus, the waiting time for a bus is included in the path cost. Notice that with a different starting time from q, e.g., $t = 06{:}01{:}00$, the path cost might be significantly different due to the changed transfer time on discrete time edges and no bus connection recorded in the corresponding schedule.

In most networks there are multiple different paths from a source to a destination. Therefore, another cost is needed to record the minimal cost of all paths between two vertices. This cost is called the "network distance" and it is defined as the cost of the shortest path from a source location to a destination location as follows.

Definition 19: Network distance

The *network distance* $d(l_s, l_d, t)$, from a source location l_s, to a destination location l_d, with t being the departure time at l_s (the arrival time at l_d), is defined as the minimum of all path costs from l_s to l_d with departure time t at l_s (arrival time t at l_d). If such a path does not exists the network distance is defined as ∞.

Since the path cost is measured in terms of transfer time, the same holds true for the network distance. The network distance therefore records the transfer time required from a source to a destination (with t being the departure time at the source/the arrival time at the destination).

With the help of the network distance it is now possible to define a reachable area. For outgoing computations, this area covers all locations that are reachable from a query point q (or covers all locations from which q is reachable for incoming computations) under the given time constraints. The following Definition 20 formalizes it.

Definition 20: Reachable area

Let N be a Multimodal Spatial Network as defined in Definition 13, G the graph representing N, Q be a set of query points with departure time (or arrival time) t, x be an edge (segment) and $d_{max} > 0$ be a maximal travel duration. Furthermore, let $\hat{d}(q, l, t) = d(q, l, t)$ be the network distance from q to l with t being the departure time at q for outgoing computations (or let $\hat{d}(q, l, t) = d(l, q, t)$ be the network distance from l to q and t is the arrival time at q for incoming computations). Then the *reachable area* $N^{ra} = (V^{ra}, E^{ra})$ is defined as the minimum and possibly disconnected accessible part that consists of a subgraph of G together with a possibly empty set of edge segments and satisfies the following conditions:

- $V^{ra} \subseteq V$,
- $\forall l((\exists x \in E^{ra}(x = (e, o_1, o_2) \wedge 0 \leq o_1 \leq o \leq o_2 \leq \lambda(e)))),$
  $\Leftrightarrow (l = (e, o) \wedge e \in E \wedge \exists q \in Q(\hat{d}(q, l, t) \leq d_{max}))$

The first condition requires the vertices of the reachable area to be a subset of the vertices of the Multimodal Spatial Network. The second condition constrains the reachable area to cover exactly those locations l with a network distance $d(l, q, t)$ from (or to) its closest $q \in Q$ that is smaller than or equal to d_{max}. The usage of edge segments in E^{ra} is used to represent partially reachable edges. Whenever an edge e is entirely covered by an isochrone, e instead of $(e, 0, \lambda(e))$ is used.

Depending on the transport modes of the transportation systems used the reachable area can have various properties. For spatial networks that allow traveling solely on continuous space edges ($\mu =$ "*csct*" or $\mu =$ "*csdt*") the area will exist of a single connected part only. As soon as any path contains one or more discrete space edges ($\mu =$ "*dsct*" or $\mu =$ "*dsdt*") the resulting reachable area is possibly, but not necessarily, disconnected.

In Figure 3.10, the parts in bold represent the reachable area for a maximal travel duration $d_{max} = 5\,\text{min}$ and a departure time $t =$ 06:06:00 from q. The numbers in parentheses are the network distance from q for every vertex. Edges close to q are entirely reachable, whereas edges further away are only partially reachable. Partially reachable edges are labeled with the offset of the reachable portion from the edge's start vertex and are modeled with the help of edge segments. For instance, on edge (v_0, v_1) only locations within an offset of 160 meters are reachable from q within the given time constraints.

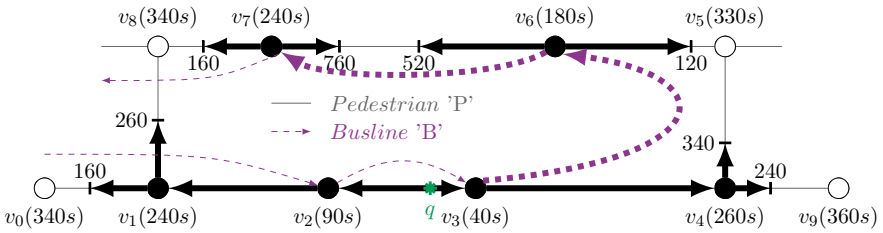


Figure 3.10.: Reachable area (d_{max}=5min, s=4m/s and $t =$ 06:01:00)

More formally, the reachable area is represented by the following set of vertices and edges:

$$
\begin{aligned}
V^{ra} &= \{v_3, v_2, v_6, v_1, v_7, v_4\},\\
E^{ra} &= \{((v_0, v_1), 160, 400), ((v_8, v_1), 260, 500), ((v_2, v_1), 360, 600),\\
&\quad ((v_1, v_2), 0, 600), ((v_3, v_2), 0, 520),\\
&\quad ((v_2, v_3), 0, 520), ((v_4, v_3), 0, 880),\\
&\quad ((v_3, v_4), 720, 880), ((v_5, v_4), 340, 500), ((v_9, v_4), 240, 400),\\
&\quad ((v_5, v_6), 120, 600), ((v_7, v_6), 520, 1000),\\
&\quad ((v_6, v_7), 720, 1000), ((v_8, v_7), 160, 400)\}.
\end{aligned}
$$

An isochrone in a Multimodal Spatial Network is defined on top of a reachable area within the same network. It is the borderline that separates the possible disconnected reachable area from the rest of the network. Its formalization is given in Definition 21.

Definition 21: Isochrone

An *isochrone* is the set of lines that separate the reachable area N^{ra} from the rest of the network. The borders contain all the locations that are reachable from the nearest query point q (all the locations from which the nearest query point q is reachable).

The isochrone can be computed from a reachable area with several algorithms. Besides alpha shape and concave hull algorithms, approaches have been discussed that use the edges from the graph rather than the vertices [73]. An isochrone can also be created by shrinking a rectangle of the size of the entire network as long as edges from the reachable area N^{ra} are obtained and no further contraction is possible. The needed operation is well known as "erodation" in the field of morphological image processing.

3.3. Real-world Networks

In the previous section only a very simple Multimodal Spatial Network has been used. In fact most networks created from geographical data are of multimodal nature. Even the transportation network shown in Figure 3.7 contains more than one modality. Since this network illustrates a region within the center of the city of Innsbruck, the capital city of the area of Tyrol, located in Austria, it is also created from geographical data. Adding information about the public transportation system makes it a true Multimodal Spatial Network.

Figure 3.11 shows a Multimodal Spatial Network with two transportation systems, $\Theta = \{\text{'P'}, \text{'B'}\}$, representing the pedestrian network with mode $\mu(\text{'P'}) =$ "*csct*" and bus network 'B' containing two routes 'A' and 'C' with mode $\mu(\text{'B'}) =$ "*dsdt*", respectively. Solid lines are segments of the pedestrian network. Dashed violet lines represent edges of the bus network. For reasons of simplicity the weights and lengths of the edges are not plotted.

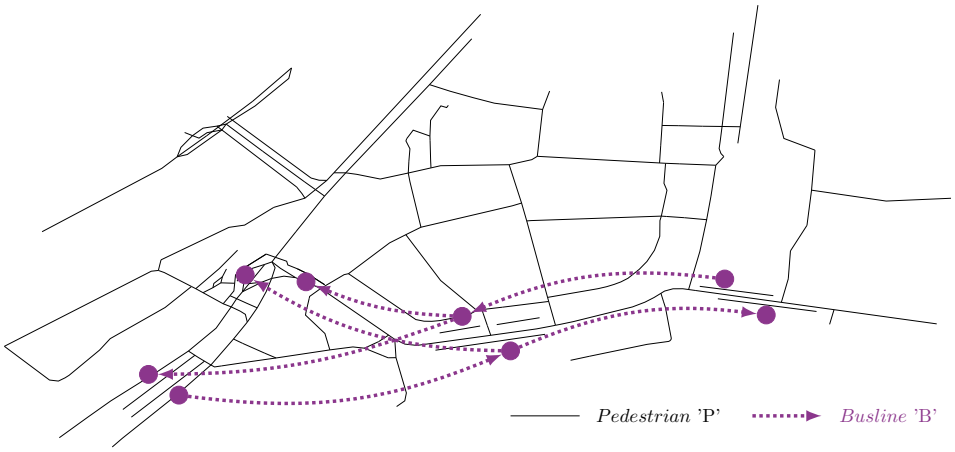


Figure 3.11.: Multimodal Spatial Network modeling the city center of Innsbruck.

As already mentioned in Section 2, many applications use OpenStreetMap (OSM) to acquire data about real-world networks. If information about the public transportation system, like schedules and locations of stations, are merged with data representing the road network from OSM, then a Multi-modal Spatial Network can be created for various regions around the globe. In theory this makes it possible to compare reachability across various cities, countries or even continents.

Although this seems straight forward there are multiple pitfalls. First, there is no official public transportation system in every city. As soon as urban areas get smaller, e.g. a village, this issue might be the main problem. Even if there is such a transportation system, there is no guarantee that information about it is usable. Data about the road network or the public transportation system might not be digitized, the digital format used might not be suitable, the data quality might be poor or the usage of the information might not be allowed due to some restriction. At the time of writing, this is a big issue for public transportation systems around areas in the middle of Europe and Africa, but totally different for regions in the United States of America. Bigger cities in the U.S. provide data about their public transportation system as files implementing the General Transit Feed Specification (GTFS) to describe schedule based information [2, 75].

3.3.1. Geographical Information

Even if there is data about the road network and the public transportation system available, data has to be stored in a comparable format. Although this might sound simply, when it comes to geographical data this is a non trivial task. The earth is often thought of as a sphere. A sphere models the reality better than a flat surface, but in geographical terms this model is still far from ideal and not exact. Because of this, the earth is modeled as a geoid. For this purpose, multiple reference systems, have been established around different areas of the world. A single reference system would be much too complicated to be exact. The needed conversions make it hard to compare coordinates exactly, since there is a need to convert the compared coordinates in a uniform projection. Some standards use the terms Spatial Reference System (SRS) or Coordinate Reference System (CRS) to refer to a specific reference system, although they can be further classified into unprojected systems and projected ones.

A well-known and often used geographic coordinate system (GCS) is the World Geodetic System 1984 (WGS84) (also known as “EPSG:4326”, since it also has been specified by the European Patroleum Survey Group (EPSG)). It uses lat-

itude and longitude to define coordinates that specify points on the surface of the earth (x, y and maybe also a z coordinate). Latitude and longitude are only perfectly suitable at the equator, but get more and more inaccurate as soon as locations are near to the north or south pole. All GCS have in common that their coordinates are defined on a three dimensional surface, which makes it hard to compute or compare certain measurements, such as distances. Even on a simple sphere, the Pythagorean theorem does not apply and the distance between two points is defined as a greater circle distance, which is much harder to compute than on a two dimensional flat surface. Therefore, also projected coordinate systems (PCS) have been defined to enable easy computations. PCS are systems where coordinates representing places on the three dimensional earth have been transformed into a two dimensional flat surface. A widely used PCS is the one specified by "EPSG:3857" also referred to as web mercator. This projection is used by most companies, like Google within Google Maps, Microsoft within Bing Maps or OpenStreetMap. It transforms coordinates into a rectangular plain. However, computations need to be carried out with care to not mix up coordinates across different reference systems.

GCS and PCS are referred to with the help of a spatial reference identifier (SRID). Only with knowledge about geographical coordinates together with the SRID comparisons within spatial information is possible. All systems that deal with geographical information, have to know about SRIDs and their definition which contains rules on how to transform coordinates in different reference systems. This also holds true for geographical databases that store datasets which are used for isochrone computation within Multimodal Spatial Networks. [48]

3.3.2. Schedule Based Information

Multiple competing standards exist that are used to describe schedule based information like the information within the public transportation system. The situation has been well described by S. Kaufmann in 2014 [55]:

"In Europe, the development of data models for data exchange within and in between transit operators began in the late 1980s, resulting in the ÖPNV-Datenmodell in Germany and Cassiope [...] Both influenced the pan-european Transmodel [...] which was ultimately standardized as EN 12896:2006. Transmodel as a reference data model served as a basis for European standard implementations, such as the TransXChange standard used for bus schedules in the United Kingdom, and the Service Interface for RealTime Information (SIRI) standard [...].

Despite all these different standards, vendor-specific data models still play a prominent role when encountering route network and schedule data. On the German market, "HaCon Fahrplan-Auskunfts-System" (HAFAS) and "Dialog-gesteuertes Verkehrsmanagement- und Auskunftssystem" (DIVA) are two of the major software suites used by public transit agencies — one using a documented exchange format, the other a format with no publicly available documentation whatsoever.

Despite there never being one transit data model adopted by any world-wide regulatory body, in recent years, "General Transit Feed Specification" (GTFS) has become some kind of de-facto standard widely used within the open data community. First developed by Portland's TriMet transit agency together with Google, it is now used by Google's Transit Planner as well as a growing number of transit application by third parties."

The last three years did not change the situation that much. GTFS is still the de-facto standard and it is used by a growing number of transit agencies. For most of the cities in the U.S. schedules are available in GTFS format. Recently, also agencies within the middle of Europe start to provide data in this standard. Therefore, geographical data can be merged with information from GTFS file in order to create Multimodal Spatial Networks. Therefore, reachability can be calculated throughout various regions of the world. More information on how this can be done is given in Chapter 7 that explains a workflow creating a dataset with information taken from OpenStreetMap and GTFS files.

CHAPTER 4

Existing Algorithms for the Computation of Isochrones

In general, there are multiple various approaches available to compute routes and isochrones within graphs and different networks. This chapter focuses on already known algorithms that are capable to compute reachable areas within Multimodal Spatial Networks (MMSN) as defined in the previous Chapter 3. New approaches also able to work in MMSN are further discussed in Chapter 5. The visualization and isochrone computation itself is later explained in Chapter 8. The memory consumption and runtime comparisons within different spatial networks is part of Chapter 9.

The algorithms described throughout this thesis have many things in common. They are based on Dijkstras algorithm [23] and are not only able to compute reachable areas in unimodal, but also multimodal networks. The difference between computing a reachable area and an isochrone can be seen from the definitions in Chapter 3. An isochrone is just the borderline of a reachable area. Since many papers mix these terms, it should be noted, that although the term "isochrone computation" is used, in fact the reachable area is computed.

All the following algorithms use a network expansion strategy, but differ in the way they handle memory:

- A memory based approach (called MDijkstra) loads the entire network into memory and performs network expansion there.

- Multimodal Incremental Network Expansion (Mine) reduces the memory needed by loading data from an underlying data source, i.e. a spatial database, on demand.

- Vertex Expiration frees memory as early as possible to further reduce the computation footprint. The approach was developed by Gamper et al. and resulted in an algorithm named Multimodal Incremental Network Expansion using vertex eXpiration (MineX) [34].

- MineX has been proved to be optimal in terms of memory consumption, but introduces the need for many data source accesses. To find a hybrid approach between MDijkstra and MineX, Innerebner described an algorithm loading multiple data points at once using circular areas. This approach has been named "Multimodal Incremental Network Expansion loading Ranges using vertex eXpiration (MineRX)" [51]. The approach can also be used without vertex expiration, which is known as algorithm MineR.

Summarizing, there were five different algorithms available, before research lead to enhanced algorithms that are described in Chapter 5:

- MDijkstra
- Mine and MineX
- MineR and MineRX

Different configurations can be used by all the algorithms, e.g. if the reachable area should be computed with incoming or outgoing direction, if it should use all the modalities in the spatial network or if the computation should be performed unimodal without schedule-based data. Also the traveling speed, various speed estimation models (more about these models will be described in Chapter 6) and the starting (or for incoming direction the destination) date and time can be configured.

4.1. In-Memory Computation

The first algorithm able to compute reachable areas within Multimodal Spatial Networks was presented at the ACM GIS conference in 2008 by Bauer et al. [9]. At that time the underlying networks were referred to as "multi-modal, schedule-based transport networks", which basically means the same thing as the current definition of MMSN. Another minor difference is that the paper describes incoming isochrone computation, while during this thesis outgoing calculations are discussed. The work by Bauer et al. uses "isochrone computation" although it relates to the calculation of a reachable area. Since the algorithm is the one most similar to the original algorithm by Dijkstra, it has been named "Multimodal Dijkstra Network Expansion" (or MDijkstra) by M. Innerebner [51]. The pseudo code of MDijkstra is listed in Algorithm 1.

Algorithm 1: Algorithm MDijkstra$(q, d_{max}, s, t, \mathsf{N})$

input : $q, d_{max}, s, t, \mathsf{N}$
output : E^{ra}, V^{ra}

1 $O \leftarrow \{(v_i, \infty) \mid v_i \in V\}$;
2 $C \leftarrow \emptyset$;
// project query point to start location (if necessary)
3 **if** q *coincides with* v **then**
4 $\quad O \leftarrow (O \setminus \{(v, \infty)\}) \cup \{(v, 0)\}$; // update the duration of v in O to 0
5 **else**
6 $\quad P = \{(v, u) \mid e \in E : d_e = \min_{\forall x \in E} dist(q, x)\}$; // find edge(s) with min distance
7 $\quad$ **foreach** $e = (v, u) \in P$ **do**
8 $\quad\quad O \leftarrow (O \setminus \{(u, \infty)\}) \cup \{(u, o/s)\}$;
$\quad\quad$ // add segments from q to edge (u, v) and to start vertex on edge
9 $\quad\quad E^{ra} \leftarrow E^{ra} \cup \{((q, o), 0, \lambda(q, o)), ((u, v), \max(0, ((o/s) - d_{max})), o)\}$;
// network expansion
10 **while** $O \neq \emptyset \wedge (v, d_v) \leftarrow dequeue(O) \wedge d_v \leq d_{max}$ **do**
11 $\quad O \leftarrow O \setminus \{v\}$;
12 $\quad C \leftarrow C \cup \{v\}$;
13 $\quad$ **foreach** $e = (v, u) \in E$ **do**
14 $\quad\quad dTemp \leftarrow d_v + \tau(e, t + d_v)$; // d_v => duration from q to v
15 $\quad\quad$ **if** $u \notin C$ **then**
$\quad\quad\quad$ // set (maybe new) minimal duration d_u from q to vertex u
16 $\quad\quad\quad O \leftarrow (O \setminus \{(u, d_u)\}) \cup \{(u, \min(d_u, dTemp))\}$;
$\quad\quad$ // output continuous edge segments only
17 $\quad\quad$ **if** $\mu(\theta(e)) \in \{$"csct", "csdt"$\}$ **then**
18 $\quad\quad\quad$ **if** $dTemp \leq d_{max}$ **then**
19 $\quad\quad\quad\quad E^{ra} \leftarrow E^{ra} \cup \{(e, 0, \lambda(e))\}$; // full segment
20 $\quad\quad\quad$ **else**
21 $\quad\quad\quad\quad E^{ra} \leftarrow E^{ra} \cup \{(e, o, \lambda(e))\}$, where $d((e, o), q, t) = d_{max}$; // partial segment
22 $\quad V^{ra} \leftarrow V^{ra} \cup \{(v, d_v)\}$;

This algorithm requires the following input parameters: the query point q, the maximal duration d_{max}, the maximum travel speed s, the starting date and time t as well as the Multimodal Spatial Network N. At the beginning of the algorithm the whole network is loaded into memory. Then it uses two data stores to compute the reachability within the network. The first store, called O (or the set of open vertices), represents an ordered data structure that holds the network vertices which have not yet been expanded by the network expansion. The second store, called C (or the set of closed vertices) holds all the vertices that have already been expanded. The variable d_v represents the minimal duration from the initial query point q to vertex v and is computed for all vertices in $O \cup C$. It is also referred to as the network distance from q to v (or $d(q, v, t)$). During network expansion it is updated to guarantee that the shortest path is used to reach every vertex with minimal traveling time, making the algorithm a greedy one [19].

At the very beginning of the algorithm, the two data stores O and C are initialized. The ordered store O is initialized with all the vertices from the network, setting all vertex network distances to infinity (line 1). The store C is initialized to the empty set and will be filled on network expiration later in the algorithm (line 2). Then, the query point is mapped to a location within the network. In the simpler case the query point coincides with a vertex and all that has to be done is to update the duration for the matching vertex (line 4). In the more complex case, when q does not coincide with a vertex, the query point is projected to locations on the nearest edges to q (line 9). After the query point projection, the network expansion starts (line 10).

The vertex v with the smallest network distance is retrieved from O by calling the dequeue operation on it. The dequeue operation of the sorted set guarantees that always the vertex with the current minimal duration from the query point is used. The vertex is then removed from O and added to C. Then the edges starting at (or for incoming isochrones all edges ending in) the vertex v are retrieved. All of them are processed in a loop (line 13). For each edge $e = (u, v)$, the network distance to the adjacent vertex u is updated, if it has not already been closed (meaning not already moved to the closed set) and is smaller than the previous network distance of the vertex. If the edge is a street edge also the temporary duration $dTemp$ is computed, since partial edges could be part of the resulting reachable area. However, this is not done for discrete edges, since partial discrete edges are not possible (e.g. one can not leave a moving train between two stations). The result itself is also again only made up of vertices (line 22) and continuous edges (line 19 for fully and 21 for partial reachable edges), but not discrete edges. The reason for this is again the fact that one can not leave a discrete edge in between (e.g. when traveling by plane from New York to Vienna) and that discrete vertices are

always connected by continuous edges (e.g. an airport is always reachable by car and/or by foot). The algorithm terminates as soon as O becomes empty or the duration of the vertex retrieved from it exceeds d_{max}.

Regarding the runtime performance of the algorithm several things can be noted. The query point projection to network vertices is mandatory for MDijksta and all the algorithms discussed throughout this thesis. The time needed of this part stays the same for all of them. The main problem of MDijkstra is that it is not scalable in terms of memory, since the whole network is loaded and added to the data source O. Therefore, the initial loading time heavily depends on the network size and the memory consumption is very high, resulting in the fact that MDijkstra is not suitable for huge networks, including Multimodal Spatial Networks of large areas. The implementation of O can be done by multiple data structures. Since the network distances to (or from) vertices are regularly changed, the sorting order of the used structure has to deal with this fact. A priority queue or a binary heap therefore suitable for O, but they are not as good for C. For the latter variable only insertion and searching for containment is important. Here either using an attribute within a vertex instance makes sense (a simple boolean flag) or using a hash table that offers constant complexity $\mathcal{O}(1)$ for both operations. The complexity of the MDijkstra algorithm itself is mostly connected to the queue used, when using a binary heap it is $\mathcal{O}((m+n) * log(n))$ [76], with m being the number of edges $|E|$ within the graph and n corresponding to the number of vertices $|V|$.

M. Innerebner described three extensions and improvements regarding the MDijkstra [51] algorithm. One of them is to use an interval-based schedule management instead of an instance-based one, while another is to batch queries to get schedules. The latter approach gets all schedules regarding one vertex u for its adjacent vertices at once, and not neighbor-by-neighbor. The third suggestions is to implement the algorithm independently from the underlying data source, so that various spatial databases can be used (i.e. Oracle Spatial or PostgreSQL/PostGIS). All of these extensions have been implemented and are part of IsoMap described in Chapter 8. The results plotted in the evaluation in Chapter 9 also make use these improvements.

4.2. Incremental Expansion

To address the disadvantage of loading the entire network into main memory, three incremental approaches were developed at the Free University of Bozen-Bolzano (FUB) that are presented in the following sections.

4.2.1. Multimodal Incremental Network Expansion

The first method has been suggested by Gamper et al. [33] in 2011. The algorithm described is named "Multimodal Incremental Network Expansion" (Mine) and was presented at the International Conference on Information and Knowledge Management (CIKM).

Mine is very similar to MDijkstra as can be seen in Algorithm 2. Changes between those two algorithms are highlighted in blue. In contrast, the set O is initialized to the empty set at the very beginning of the procedure (line 1). It is first filled in the query point projection. On network expansion steps vertices that have not been explored yet, but are connected to the currently explored vertex v, are loaded from the underlying data source with an infinite network distance (line 15). As vertices get loaded as soon as they are needed (and not earlier), this approach is also referred to as "on demand" or "vertex-by-vertex" loading. Besides these two differences, the algorithm does exactly the same steps as MDijkstra. However, these changes are sufficient to drastically change the runtime behavior.

Algorithm 2: Algorithm Mine($q, d_{max}, s, t, \mathsf{N}$)

input : $q, d_{max}, s, t, \mathsf{N}$
output : E^{ra}, V^{ra}

$O \leftarrow \emptyset$;
$C \leftarrow \emptyset$;
// project query point to start location (if necessary)
if q *coincides with* v **then**
  $O \leftarrow \{(v, 0)\}$; // update the duration of v in O to 0
else
  $P = \{(v, u) \mid e \in E : d_e = \min_{\forall x \in E} dist(q, x)\ \}$; // find edge(s) with min distance
  foreach $e = (v, u) \in P$ **do**
    $O \leftarrow \{(u, o/s)\}$;
    // add segments from q to edge (u, v) and to start vertex on edge
    $E^{ra} \leftarrow E^{ra} \cup \{((q, o), 0, \lambda(q, o)), ((u, v), \max(0, ((o/s) - d_{max})), o)\}$;
// network expansion
while $O \neq \emptyset \wedge (v, d_v) \leftarrow dequeue(O) \wedge d_v \leq d_{max}$ **do**
  $O \leftarrow O \setminus \{v\}$;
  $C \leftarrow C \cup \{v\}$;
  foreach $e = (u, v) \in E$ **do**
    if $u \notin \{O \cup C\}$ **then**
      $O \leftarrow O \cup \{(u, \infty)\}$;
    $dTemp \leftarrow d_v + \tau(e, t + d_v)$; // d_v => duration from q to v
    if $u \notin C$ **then**
      // set (maybe new) minimal duration d_u from q to vertex u
      $O \leftarrow (O \setminus \{(u, d_u)\}) \cup \{(u, \min(d_u, dTemp))\}$;
    // output continuous edge segments only
    if $\mu(\theta(e)) \in \{$"csct", "csdt"$\}$ **then**
      if $dTemp \leq d_{max}$ **then**
        $E^{ra} \leftarrow E^{ra} \cup \{(e, 0, \lambda(e))\}$; // full segment
      else
        $E^{ra} \leftarrow E^{ra} \cup \{(e, o, \lambda(e))\}$, where $d((e, o), q, t) = d_{max}$; // partial segment
  $V^{ra} \leftarrow V^{ra} \cup \{(v, d_v)\}$;

If a computed reachable area is relatively small in relation the datasets size (i.e. 10min maximal duration with a maximal traveling speed of 5m/s in the city of Berlin). Mine achieves much better runtime than MDijkstra. The cause for this is simply that although more queries are sent to the underlying data source, less memory is required to handle the results. The biggest advantage of Mine is that it is able to compute reachable areas and isochrones within very large datasets that would otherwise not fit into memory. However, as traveling speed and/or maximal duration increased, Mine gets slow and there is a break-even point, where MDijkstra gets faster for all networks that fit into memory. The complexity of the algorithm is the same as for MDijkstra [51].

Although this algorithm is able to deliver results in very large networks, this does not hold true for large reachable areas. Because the vertices that have been processed still stay in memory, the limiting factor now is the size of the computed area. Vertices are added to C in line 12 and their existence is checked in line 14, but no vertex is ever removed from C. Therefore, Mine is able to work in very large networks, but not to compute very large results.

4.2.2. Optimizing the Memory Footprint

To further reduce the memory footprint of isochrone computations in Multimodal Spatial Networks, a new technique called vertex expiration was introduced by Gamper et al. in 2012 at the 24th International Conference on Scientific and Statistical Database Management (SSDBM) [34]. The Mine algorithm was improved with this technique which led to the name "Multimodal Incremental Network Expansion using vertex eXpiration" (MineX).

MineX is listed in Algorithm 3. Changes between Mine and MineX are highlighted in blue. The idea behind vertex expiration is to purge a vertex from memory as early as possible, without the need to re-load it from the underlying data source. As has been proven by Gamper et al. a closed vertex, $u \in C$, can be expired if all its neighbors are either closed or expired (depending on the direction "neighbors" refers to all the in-neighbors that are connected via an edge (v, u) or all out-neighbors that are connected via an edge (u, v)). In order to do so, each vertex is assigned with a counter that keeps track of the connected edges that have been traversed (lines 3, 7 and 14). With the help of the counter, that is initially set to the incoming vertex degree (or depending on the traveling direction the outcoming vertices) and reduced as soon as one of its neighboring vertices is expanded, it is possible to know when a vertex will not be needed anymore. As soon as this very counter reaches zero, the recording vertex can be removed from memory and also deleted from set C (lines 18 and 28).

Algorithm 3: Algorithm MineX$(q, d_{max}, s, t, \mathsf{N})$

input : $q, d_{max}, s, t, \mathsf{N}$
output : E^{ra}, V^{ra}

$O \leftarrow \emptyset$; $C \leftarrow \emptyset$;

// project query point to start location (if necessary)
if q *coincides with* v **then**
  $O \leftarrow \{(v, 0, cnt_v)\}$; // update the duration of v in O to 0
else
  $P = \{(v, u) \mid e \in E : d_e = \min_{\forall x \in E} dist(q, x)\ \}$; // find edge(s) with min distance
  foreach $e = (v, u) \in P$ **do**
    $O \leftarrow \{(u, o/s, cnt_u)\}$;
    // add segments from q to edge (u, v) and to start vertex on edge
    $E^{ra} \leftarrow E^{ra} \cup \{((q, o), 0, \lambda(q, o)), ((u, v), \max(0, ((o/s) - d_{max})), o)\}$;

// network expansion
while $O \neq \emptyset \wedge (v, d_v, cnt_v) \leftarrow dequeue(O) \wedge d_v \leq d_{max}$ **do**
  $O \leftarrow O \setminus \{v\}$;
  $C \leftarrow C \cup \{v\}$;
  foreach $e = (u, v) \in E$ **do**
    if $u \notin \{O \cup C\}$ **then**
      $O \leftarrow O \cup \{(u, \infty, cnt_u)\}$;
    $dTemp \leftarrow d_v + \tau(e, t + d_v)$; // d_v => duration from q to v
    if $u \notin C$ **then**
      // set (maybe new) minimal duration d_u from q to vertex u
      $O \leftarrow (O \setminus \{(u, d_u)\}) \cup \{(u, \min(d_u, dTemp))\}$;
    // decrease counter of adjacent vertices (and expire them if possible)
    $cnt \leftarrow cnt_u - 1$; // $cntu$ refers to the current counter value of vertexs u
    if $u \in C \wedge cnt = 0$ **then**
      $C \leftarrow C \setminus \{u\}$;
    else
      $O \leftarrow (O \setminus \{u, d_u, cnt_u\}) \cup \{u, d_u, cnt\}$;
    // output continuous edge segments only
    if $\mu(\theta(e)) \in \{$"csct", "csdt"$\}$ **then**
      if $dTemp \leq d_{max}$ **then**
        $E^{ra} \leftarrow E^{ra} \cup \{(e, 0, \lambda(e))\}$; // full segment
      else
        $E^{ra} \leftarrow E^{ra} \cup \{(e, o, \lambda(e))\}$, where $d((e, o), q, t) = d_{max}$; // partial segment
  // decrease counter of expanded vertex (and expire it if possible)
  $cnt \leftarrow cnt_v - 1$;
  if $v \in C \wedge cnt = 0$ **then**
    $C \leftarrow C \setminus \{v\}$;
  else
    $O \leftarrow (O \setminus \{v, d_v, cnt_v\}) \cup \{v, d_v, cnt\}$;
  $V^{ra} \leftarrow V^{ra} \cup \{(v, d_v)\}$;

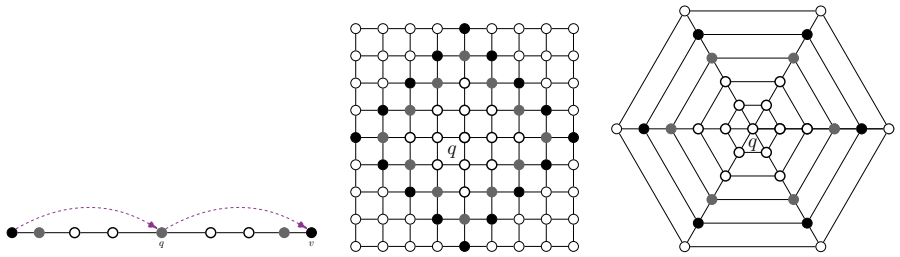
The most notable property of MineX and its vertex expiration is that it is optimal in the sense that only portions of the network are loaded that will become part of the isochrone, and each edge is loaded only once [34].

With the help of the two sets O and C and the counter introduced by the vertex expansion, each vertex can be assigned to one of three disjoint groups:

- open vertices (O)
- closed vertices (C)
- expired vertices (X)

A vertex in the O group is part of the O set and has not been expanded yet. In the group C, vertices that have already been expanded, are kept as long as their counter has a value greater than zero. It is important that they are not removed from memory to prevent cyclic expansions or loading the same vertex multiple times from the underlying data source. As soon as a vertex decreases its counter to zero, it is expired and becomes part of the X set, which is not kept in memory.

Although it has been shown that vertices can not be expired according to a least recently used (LRU) strategy [34], most Multimodal Spatial Networks created from geographic data are of similar structure to a spider or grid network. In these networks vertex expiration typically results in lines that exist of the various vertex groups: vertices contained in the open group at the very outside of each reachable area, closed but not yet expanded vertices just a layer afterwards, while most of the remaining vertices belong to the expired group and form the center of the reachable area. This behavior is illustrated in Figure 4.1 and Figure 4.2. It shows open vertices from the O group in black, closed ones from C in gray and expired data from the X group in white. For Figure 4.1 the edges are plotted without direction (so they can be passed in both directions). For the LRU same the dotted edge refers to a discrete bus edge.



(a) LRU (b) Grid (c) Spider

Figure 4.1.: Expiration in synthetic networks.

(a) Bolzano (b) Innsbruck

Figure 4.2.: Expiration in real-world networks.

The complexity of the MineX algorithm stays the same as the one for Mine and for MDijkstra. However, the memory consumption is less and so MineX is able to compute reachable areas in very large networks even when the resulting area is huge. The main disadvantage of MineX is that it still needs to access the underlying data source a lot of times (for each vertex at least one time, if schedules of public transportation systems are involved, sometimes even more often).

4.2.3. Loading Data using Ranges

To address the disadvantages of the previous Mine and MineX algorithms, hybrid approaches between MDijkstra and these algorithms have been described [51]. The proposed approach performs initial network loading and expansion just like Mine(X) does, but reduces the number of calls to the underlying data source, by loading the network in portions. These portions (also referred to as chunks) load multiple vertices at once. As soon as a vertex is retrieved from the network, circular ranges around the queried vertex are created that are used for loading. These loading ranges are defined by a center, which corresponds to the queried vertex, and a radius that is bounded by the remaining travel time available. The algorithm then utilizes spatial database functionality to load all the vertices within the circular range. Already loaded information are stored in memory, so that during network expansion only data not already loaded before, produce the need to communicate with the data source. Besides the different loading approach using chunks, the algorithms network expansion is identical to the one of Mine(X). Since circular ranges are used for loading the algorithm is named “Multimodal Incremental Network Expansion loading Ranges (using vertex eXpiration)” or MineR(X). It is given in pseudo code in Algorithm 4, while changes between MineX and MineRX are highlighted in blue.

Algorithm 4: Algorithm MineRX($q, d_{max}, s, t, \mathsf{N}$)

input : $q, d_{max}, s, t, \mathsf{N}$
output : E^{ra}, V^{ra}

1 $O \leftarrow \emptyset$; $C \leftarrow \emptyset$; $Q \leftarrow \emptyset$; $R \leftarrow \emptyset$; $T \leftarrow \emptyset$;

// project query point to start location (if necessary)
2 **if** q *coincides with* v **then**
3 $\quad O \leftarrow \{(v, 0, cnt_v)\}$; // update the duration of v in O to 0
4 **else**
5 $\quad P = \{(v,u) \mid e \in E : d_e = \min_{\forall x \in E} dist(q,x)\}$; // find edge(s) with min distance
6 $\quad$ **foreach** $e = (v,u) \in P$ **do**
7 $\quad\quad O \leftarrow \{(u, o/s, cnt_u)\}$;
$\quad\quad$ // add segments from q to edge (u,v) and to start vertex on edge
8 $\quad\quad E^{ra} \leftarrow E^{ra} \cup \{((q,o), 0, \lambda(q,o)), ((u,v), \max(0, ((o/s) - d_{max})), o)\}$;

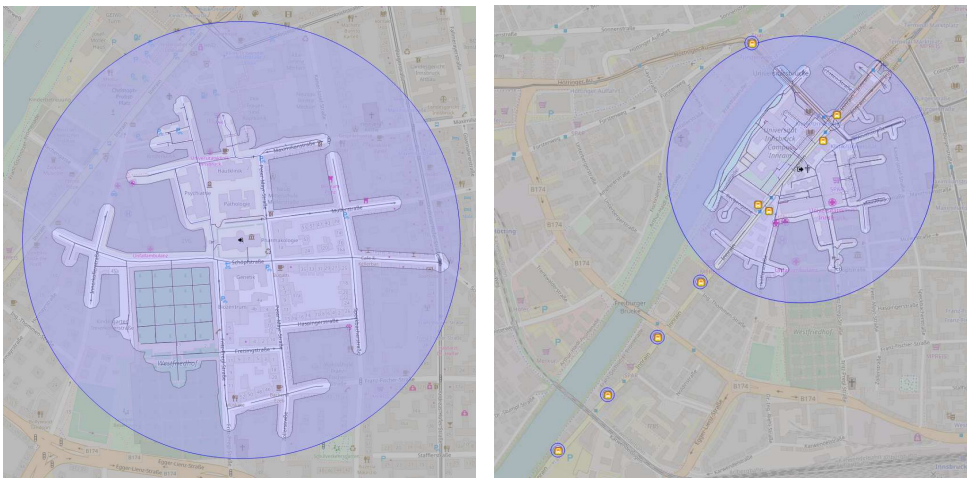
// network expansion
9 **while** $O \neq \emptyset \wedge (v, d_v, cnt_v) \leftarrow dequeue(O) \wedge d_v \leq d_{max}$ **do**
10 $\quad O \leftarrow O \setminus \{v\}$;
11 $\quad C \leftarrow C \cup \{v\}$;
$\quad$ // load vertices by using a range query
12 $\quad$ **if** $degree(v) = 0$ **then** // v is seed
$\quad\quad$ // create circle with center v and radius r_v...
13 $\quad\quad r_v \leftarrow (d_{max} - d_v) * s$;
14 $\quad\quad T \leftarrow \text{makeCircle}(v, r_v)$;
15 $\quad\quad$ **foreach** $(v', r_{v'}) \in R$ **do**
$\quad\quad\quad$ // ...and subtract previous ranges to prevent loading v multiple times
16 $\quad\quad\quad$ **if** $d_e(v, v') < r_v + r_{v'}$ **then**
17 $\quad\quad\quad\quad T \leftarrow T \setminus R \cap \rho(v', r_{v'})$;
18 $\quad\quad R \leftarrow R \cup \{(v, r_v)\}$;
19 $\quad\quad Q \leftarrow queryInRange(T)$; // issue range query using circular range T
20 $\quad\quad$ **foreach** $(u', v') \in Q$ **do**
21 $\quad\quad\quad$ **if** $u' \notin \{O \cup C\}$ **then**
22 $\quad\quad\quad\quad O \leftarrow O \cup \{(u', \infty, cnt_{u'})\}$;
23 $\quad\quad\quad$ **if** $v' \notin \{O \cup C\}$ **then**
24 $\quad\quad\quad\quad O \leftarrow O \cup \{(v', \infty, cnt_{v'})\}$;

$\quad$ // continue expansion like MineX would (starting from line 12)
25 $\quad$ **foreach** $e = (u, v) \in E$ **do**
26 $\quad\quad$ **if** $u \notin \{O \cup C\}$ **then**
27 $\quad\quad\quad O \leftarrow O \cup \{(u, \infty, cnt_u)\}$;
28 $\quad\quad dTemp \leftarrow d_v + \tau(e, t + d_v)$;
29 $\quad\quad$ **if** $u \notin C$ **then**
30 $\quad\quad\quad O \leftarrow (O \setminus \{(u, d_u)\}) \cup \{(u, \min(d_u, dTemp))\}$;
31 $\quad\quad cnt \leftarrow cnt_u - 1$;
32 $\quad\quad$ **if** $u \in C \wedge cnt = 0$ **then**
33 $\quad\quad\quad C \leftarrow C \setminus \{u\}$;
34 $\quad\quad$ **else**
35 $\quad\quad\quad O \leftarrow (O \setminus \{u, d_u, cnt_u\}) \cup \{u, d_u, cnt\}$;
36 $\quad\quad$ **if** $\mu(\theta(e)) \in \{$"csct", "csdt"$\}$ **then**
37 $\quad\quad\quad$ **if** $dTemp \leq d_{max}$ **then**
38 $\quad\quad\quad\quad E^{ra} \leftarrow E^{ra} \cup \{(e, 0, \lambda(e))\}$;
39 $\quad\quad\quad$ **else**
40 $\quad\quad\quad\quad E^{ra} \leftarrow E^{ra} \cup \{(e, o, \lambda(e))\}$, where $d((e,o), q, t) = d_{max}$;
41 $\quad cnt \leftarrow cnt_v - 1$;
42 $\quad$ **if** $v \in C \wedge cnt = 0$ **then**
43 $\quad\quad C \leftarrow C \setminus \{v\}$;
44 $\quad$ **else**
45 $\quad\quad O \leftarrow (O \setminus \{v, d_v, cnt_v\}) \cup \{v, d_v, cnt\}$;
46 $\quad V^{ra} \leftarrow V^{ra} \cup \{(v, d_v)\}$;

The set R holds the centers of all issued range queries. The information from R is then used to subtract overlapping parts with previously performed queries (line 17). For this a newly created circular range is first stored in set T. From it overlaps are subtracted with the help of R. Then the range recorded in T is used to issue a range query (which is not necessarily circular anymore, since parts have been cut out) in line 19. The set Q is used to hold the resulting vertices from the last range query (line 19. The vertices in O are then added to the open set O (after double-checking if parts have already been loaded). The rest of the algorithm works similar to MineX.

Using ranges during network expansion has multiple advantages. In general, circular shapes reflect a reachable area quite well. This is especially true if mostly continuous parts of the network are used. In the best case, when only continuous edges are traversed, only a single range query is needed to load all the vertices reachable. The radius of the circle can be easily determined by the traveling speed and the remaining time available (at the very beginning equal to maximal travel duration d_{max}), i.e. if a person starts from query point q with a maximal traveling speed of 5 km/h and a maximal duration of 5 minutes, then the circle covering all vertices reachable has a radius of approximately 417 meters, since $v = (s/t) => s = v * t = 5[km/h] * 5[min] = (5000[m]/3600[s]) * 300[s] => s = 416,\overline{6}[m]$.

As soon as discrete edges need to be considered, mostly always multiple queries are needed to compute the reachable area within the network. The reason for this is, that a discrete edge involves schedules that are able to reach transportation stations most probably not part of the circular ranges already loaded. Since schedules are defined by time and not speed, the upper bound of the circle's radius is not applicable for discrete edges. However, the number of queries to the underlying data source needed is less when compared to Mine and MineX. Figure 4.3(a) shows a unimodal query without discrete edges where all vertices are loaded within one query, while Figure 4.3(b) shows a query including multiple modalities and discrete edges, that result in multiple queries.



(a) Unimodal range loading (b) Multimodal range loading

Figure 4.3.: MineR(X) query ranges in Innsbruck ($travelspeed = 4.5km/h$ and $d_{max} = 5min$).

To determine if an expanded vertex has already been loaded by a previous range or if it was used to trigger a new query, its degree is significant (line 19). If the reachability is computed for outgoing direction, then the out-degree is used, while for incoming directions, the in-degree is of importance. If the degree of interest is zero, then at least one of the surrounding vertices has not been loaded from the data source before. Therefore, another request to the data source is needed (line 12 and the vertex with $degree = 0$ becomes a seeding vertex (or simply a seed) for the next range query. To prevent vertices from being loaded multiple times, i.e. when a vertex v is positioned in the middle of two previously loaded seeds with a radius big enough to cover v, already loaded ranges are subtracted from the circles created by new seeds (line 17). To enable this behavior all seeds are kept in a sorted set R together with the radii used (line 18). After this subtraction process, the remaining circular range is queried (line 19). Loading ranges this way incorporates the data source itself, since it has to be able to deliver all vertices in a specific range. Therefore, a spatial database is needed when using MineR(X) in contrast to the algorithms before. The database has to implement geometric functions to build circles and operators to check for inclusion within a geometric shape. This can be ensured by checking for support of the "OpenGIS Implementation Specification for Geographic information - Simple feature access" [91], the "SQL/MM specification" [118] or similar specifications.

Loading vertices in chunks has several effects for MineR(X). Since information about a vertex is only a few bytes in size, loading multiple at once fills up transferred block sizes better. When loading vertex-by-vertex at least one

block is needed. Depending on the hardware used the block size transferred from a storage holder (a database or a file on a drive) may vary, but usually is more than only a few bytes. For hard disk drives a block size typically is 512 bytes, for specialized drives even 4 kilobytes, so querying for a vertex is not efficient. Transferring hundreds or thousands of vertices utilizes block size a lot better. When it comes to disk-locality chunks are advantageous as well. If the data source holding the vertices is clustered in the way that geographically near points are sequentially stored, the access to the data is more or less sequential and not single queries having the nature of random (disk) accesses. Sequential reading reduces I/O costs and utilizes caching mechanisms of the involved storage devices better.

Another advantage of loading ranges is that their size can be optimized to fit a special purpose. While systems holding a lot of memory allow for loading of huge circles containing thousands of vertices, systems with only a few bytes of main memory may choose to load only ranges with hundreds of data points, even when the traveling duration and/or speed is high. Constraints can not only be applied to single ranges, but to the whole algorithm as well. Decreasing the radius of the loaded ranges not solely by the remaining travel duration, but also with a maximal amount of vertices permitted to be in memory, allows to reduce the algorithms memory footprint and makes it better suitable for systems with small memory. However, in the worst case MineR(X) behaves like Mine(X) and loads each vertex with a single query.

To allow limitation by memory, the algorithm has to know about the number of vertices that would be loaded by a certain range query. Therefore, some additional precomputations are needed when using constraints. In particular, the number density needs to be computed for every vertex that can trigger the loading of a range and become its center. The number density is defined as the number of specified objects per unit. The considered unit refers to the circular range while the specified object corresponds to a vertex. The result of the precomputing looks like Table 4.1 and its illustration in Figure 4.4.

Vertex	Density	Radius
v_1	100	100
v_1	200	165
v_1	300	230
v_1	400	280
v_1	500	340
v_1	600	420
⋮	⋮	⋮
v_{123}	100	300
v_{123}	200	600
v_{123}	300	750
v_{123}	400	800
v_{123}	500	880
v_{123}	600	960
⋮	⋮	⋮

Table 4.1.: Precomputed vertex number densities.

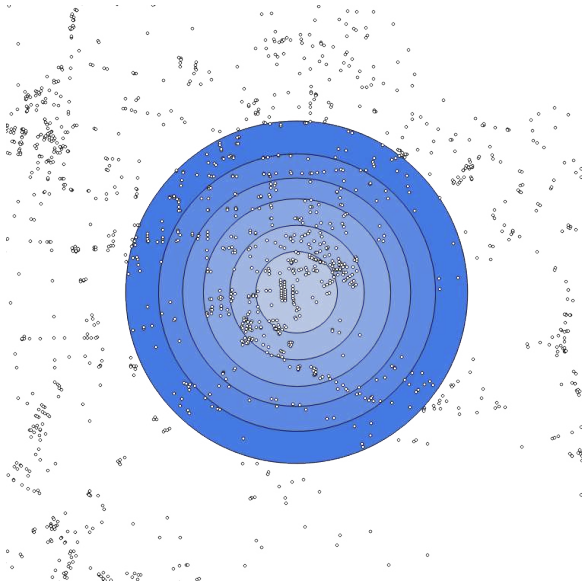


Figure 4.4.: Precomputation of radii for vertex number densities.

With the knowledge about the radii when using circular ranges, e.g. vertex v_1 loads exactly 200 vertices when using a radius of 165 meters, it is possible to always load the same amount of data. Using interpolation allows to use unknown density values, e.g. when loading 250 vertices. However, there remain two major problems with this precalculation step. At first, it is computational intense. Although optimization techniques have been applied to this precalculation [51], it still takes up to multiple hours for a dataset representing a country when only using ten different density values with a maximal density of only 1000. Second and even more problematic is the selection of good values for the density. While higher values increase computation time even more and might not be meaningful for some areas, values too low will only be helpful for systems with very small block sizes or tiny amounts of memory. High values in rural areas might range up to the nearest congested area, while in dense places low values will only be distinguishable within some small changes in radii.

Besides problems with the precomputation there are several disadvantages introduced by MineR(X). First, it builds upon geometric operations. As a result, a spatial database supporting those kind of operations is a hard requirement and the need to correctly deal with coordinate reference systems adds to the algorithms complexity. Transforming coordinates adds overhead and effects the runtime of the computation. Although the number of queries issued is

smaller when compared to Mine(X), every single query is more complex. Handling intersecting ranges again adds to the algorithms runtime. Using circles also has a negative effect on database index operation. Spatial data is indexed with the help of its minimum bounding rectangle, also known as the bounding box of the geometric data. Database operators therefore ensure containment of a point or vertex in another shape, by a two step procedure. First, the bounding boxes of the shape and point are compared by using an index operator [92]. After that additional mathematical operations, like distance computation to the center of the shape and comparison with its radius, might need to be performed to check for containment within or intersection with the shape. Using circular ranges can not only be based on index operations, since it always needs additional checks. Each intersecting circle adds one additional check to every query.

Loading data in ranges leads to some problems regarding memory consumption. At first, the ranges have to be kept in memory to prevent information from being loaded multiple times. Second, the vertices loaded by a range are not always reachable by edges and will be kept in the set O of the algorithm until the computation is finished (such vertices are called false positive loaded ones). False positives are never being released from memory by a possible enabled vertex expiration mechanism. Besides reserving memory this behavior also has bad effects on computation time. For programming languages using garbage collection, like Java or Python, these vertices have to be checked if they can be freed within a garbage collection process and therefore, the false positively loaded vertices also reduce computation performance.

CHAPTER 5

Extending and Improving Isochrone Algorithms

While in Chapter 4 algorithms have been discussed, which were developed at the Free University of Bozen-Bolzano (FUB), now algorithms introduced by the author of the thesis at hand, are presented. Enhancements regarding the isochrone application capabilities that have been established by the author, are later given in Chapter 6.

In the first section, findings are presented that improve computation runtime in general. The performed changes have been applied to all the algorithms presented throughout this thesis. Then, three newly developed algorithms are explained that address shortcomings and target to reduce the time needed to compute isochrones. They are termed in the same way as the algorithms from the previous Chapter 4:

- *Algorithm MineRB* is a variation of the MineR postboning and batching multiple circular ranges as long as possible to further reduce the communication with the data source. The variation has been named "Multimodal Incremental Network Expansion loading Ranges in Batches" (MineRB). It is described in Section 5.2.

- *Algorithms MineT and MineTX*: The circular ranges used by MineR(X) seem to be optimal in terms of reachability by foot, but are difficult to compute. Spatial databases use rectangular shapes, so called bounding boxes, to index geometrical and geographical data. Therefore, loading rectangular shapes allows to directly use the database indices. An approach using rectangles for loading (so called tiles) has been termed "Multimodal Incremental Network Expansion loading Tiles (using vertex eXpiration)" (MineT(X)) and has been published by Krismer et al. at the 29th International Conference on Scientific and Statistical Database Management (SSDBM 2017) [65]. The two algorithms MineT and MineTX are explained in Section 5.3.

In Section 5.4, a method is explained that allows to compute isochrones based on previously calculated results. This addresses the problem that all the algorithms always start the calculation from scratch without the knowledge about previously performed computations. The approach has been named "incremental calculation".

5.1. Network Expansion Revisited

The network expansion is a very important part of isochrone computation, since it applies to all the algorithms. Therefore, it has been studied in detail, including profiling techniques that lead to new insights. These are presented in this section.

As mentioned before, there are two data structures used during network expansion. The set of loaded vertices O and the set C of already expanded vertices. Since the order of vertices in O is changed throughout computation, such that the vertex that will be expanded next is the one with the shortest duration to the initial query point q, the data structure needs to be optimized for such re-orderings. There are multiple ways to optimize for this. One thing that should be prevented is re-sorting the structure after every change. This takes quit some time and is suboptimal in terms of performance. Another possibility is to remove the vertex with the old duration and to re-add it with the updated one. This leads to two much simpler operations. It can be easily seen that a simple data structure, like a sorted array or even a sorted list, is not optimal for this purpose, since both operations would be of complexity $\mathcal{O}(n)$. A tree could do both operations in $\mathcal{O}(log(n))$, providing a much better solution. However, a tree would be slower, when dequeueing from the structure (lists and arrays would have a complexity of $\mathcal{O}(1)$, while a tree sticks to $\mathcal{O}(log(n))$). A much better data structure to use is a heap or a priority

queue that is backed by a heap. By taking advantage of the fact that network distances to the vertices d is always greater or equal to zero and that it is only decreased, but never increased, an optimized implementation can be found. M. Freedman and R. Tarjan developed a specific data structure exactly for this purpose in 1987 at the University of California in San Diego. They named it Fibonacci Heap and published it in [32]. Another well-suited data structure, that for the implementation of Dijkstras algorithm according to Cormen [19] is even more favorable, is a binary heap. When using a fibonacci heap, the complexity is $\mathcal{O}(m + n * log(n))$ instead of $\mathcal{O}((m + n) * log(n))$ when using a binary heap [76], with m being the number of edges $|E|$ within the graph and n corresponding to the number of vertices $|V|$.

Depending on the actual heap implementation used the runtime can vary a lot, e.g. the Priority Queue implementation in the Java JDK is backed by a balanced binary heap, but the elements in it are not directly updateable (meaning they have to be removed and re-inserted to perform an update). Structures, allowing direct updates of the elements tracked, perform much better. Regarding this type of structures an implementation of a binary heap and the FibonacciHeap implementation by Keith Schwarz from the computer science department at the University of Stanford [111] are available. A comparison of these structures is given during Chapter 9 in Section 9.3.

Many additional improvements have been realized as well. Utilizing faster data structures in general and re-writing database queries to better use indices are just some of the tasks that have been done. A complete list of changes regarding speed-improvements and the correction of different defects is huge. However, all changes are part of a changelog file that is kept in the version control system available at [61].

5.2. Batching Multiple Database Requests

One idea to further reduce the number of request to the data source is to postbone queries as long as possible. Doing this multiple ranges can be joined, resulting in a reduced need for communication between the algorithm and the spatial database. An implementation of this approach was developed in 2017 at the University in Innsbruck. It has been named “Multimodal Incremental Network Expansion loading Ranges in Batches” (MineRB).

Before any range is loaded from the database, the network expansion is continued as long as possible, without transferring any information from the data source. Vertices that would trigger a query, i.e. the ones with a degree equal

to zero, are stored in a separate set, the so called seed cache. Only after all the available vertices from the priority queue have been expanded, the cached seeds are used to load all the ranges within a single range query. This procedure violates one of the requirements that the Dijkstra algorithm builds on, namely the sorting of the visited vertices according to the duration from the query point, that is guaranteed by the priority queue. Hence, parts of the expansion are performed multiple times and therefore more often when compared to the MDijkstra, Mine(X) and MineR(X) algorithms. As a result, MineRB is a trade-off between the number of database queries and the computation time in memory. Another effect is, that vertex expiration is not applicable anymore, since it also builds on the processing order of the vertices in the priority queue.

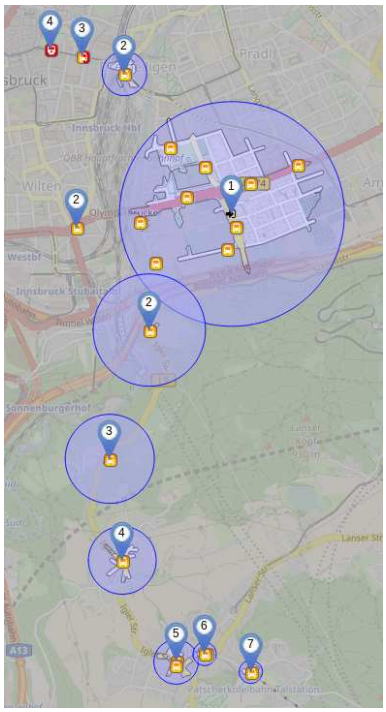


Figure 5.1.: Postboned range queries batch loaded in the city of Innsbruck.

A sample output of this algorithm for the city of Innsbruck can be seen in Figure 5.1. Although eleven ranges need to be loaded from the spatial database, only seven queries are needed. Each range in the sample is highlighted with a numbered marker which reflects the time of the query.

Especially if new ranges overlap old ones, there is a need to process the same part of the network multiple times, because new routes to an already processed vertex become available. If that is the case and the new route reduces the overall duration to reach an already processed vertex, potentially big parts of the network have to be re-calculated, increasing the overall computation time. However, for some systems loading in postboned batches can lead to a better performance, since typically the communication with the data source is done via a network (to a dedicated database server). Due to the missing vertex expiration the system has to be able to keep the whole computed reachable area in memory, but then a fast processor might be able to diminish the problem of calculating portions of the network multiple times and benefit from the reduced data source accesses, resulting in a faster overall performance. MineRB is listed in Algorithm 5, while differences to MineR are highlighted in blue.

Algorithm 5: Algorithm MineRB($q, d_{max}, s, t, \mathsf{N}$)

input : $q, d_{max}, s, t, \mathsf{N}$
output : E^{ra}, V^{ra}

1 $O \leftarrow \emptyset$; $C \leftarrow \emptyset$; $Q \leftarrow \emptyset$; $R \leftarrow \emptyset$; $T \leftarrow \emptyset$; $B \leftarrow \emptyset$;

// project query point to start location (if necessary)
2 **if** q *coincides with* v **then**
3 $\quad O \leftarrow \{(v, 0)\}$; // update the duration of v in O to 0
4 **else**
5 $\quad P = \{(v, u) \mid e \in E : d_e = \min_{\forall x \in E} dist(q, x)\}$; // find edge(s) with min distance
6 $\quad$ **foreach** $e = (v, u) \in P$ **do**
7 $\quad\quad O \leftarrow \{(u, o/s)\}$;
$\quad\quad$ // add segments from q to edge (u, v) and to start vertex on edge
8 $\quad\quad E^{ra} \leftarrow E^{ra} \cup \{((q, o), 0, \lambda(q, o)), ((u, v), \max(0, ((o/s) - d_{max})), o)\}$;

// network expansion
9 **while** $O \neq \emptyset \wedge (v, d_v) \leftarrow dequeue(O) \wedge d_v \leq d_{max}$ **do**
10 $\quad O \leftarrow O \setminus \{v\}$;
11 $\quad C \leftarrow C \cup \{v\}$;
12 $\quad$ **if** $degree(v) = 0$ **then** // v is seed
13 $\quad\quad B \leftarrow B \cup \{v\}$;

$\quad$ // load vertices by using a range query
14 $\quad$ **if** $O = \emptyset$ **then** // v is seed
15 $\quad\quad T \leftarrow \emptyset$;
16 $\quad\quad$ **foreach** $b \in B$ **do**
$\quad\quad\quad$ // create circle with center b and radius r_b...
17 $\quad\quad\quad r_b \leftarrow (d_{max} - d_b) * s$;
18 $\quad\quad\quad T \leftarrow T \cup makeCircle(b, r_b)$;
19 $\quad\quad$ **foreach** $(v', r_{v'}) \in R$ **do**
$\quad\quad\quad$ // ...and subtract previous ranges to prevent loading v multiple times
20 $\quad\quad\quad$ **if** $d_\epsilon(v, v') < r_v + r_{v'}$ **then**
21 $\quad\quad\quad\quad T \leftarrow T \setminus R \cap \rho(v', r_{v'})$;
22 $\quad\quad$ **foreach** $b \in B$ **do**
23 $\quad\quad\quad R \leftarrow R \cup \{(b, r_b)\}$;
24 $\quad\quad B \leftarrow \emptyset$; // reset batch set B to empty set
25 $\quad\quad Q \leftarrow queryInRange(T)$; // issue range query using circular range T
26 $\quad\quad$ **foreach** $(u', v') \in Q$ **do**
27 $\quad\quad\quad$ **if** $u' \notin \{O \cup C\}$ **then**
28 $\quad\quad\quad\quad O \leftarrow O \cup \{(u', \infty)\}$;
29 $\quad\quad\quad$ **if** $v' \notin \{O \cup C\}$ **then**
30 $\quad\quad\quad\quad O \leftarrow O \cup \{(v', \infty)\}$;

$\quad$ // continue expansion like Mine would (starting from line 13)
31 $\quad$ **foreach** $e = (u, v) \in E$ **do**
32 $\quad\quad$ **if** $u \notin \{O \cup C\}$ **then**
33 $\quad\quad\quad O \leftarrow O \cup \{(u, \infty)\}$;
34 $\quad\quad dTemp \leftarrow d_v + \tau(e, t + d_v)$;
35 $\quad\quad$ **if** $u \notin C$ **then**
36 $\quad\quad\quad O \leftarrow (O \setminus \{(u, d_u)\}) \cup \{(u, \min(d_u, dTemp))\}$;
37 $\quad\quad$ **if** $\mu(\theta(e)) \in \{$"csct", "csdt"$\}$ **then**
38 $\quad\quad\quad$ **if** $dTemp \leq d_{max}$ **then**
39 $\quad\quad\quad\quad E^{ra} \leftarrow E^{ra} \cup \{(e, 0, \lambda(e))\}$;
40 $\quad\quad\quad$ **else**
41 $\quad\quad\quad\quad E^{ra} \leftarrow E^{ra} \cup \{(e, o, \lambda(e))\}$, where $d((e, o), q, t) = d_{max}$;
42 $\quad V^{ra} \leftarrow V^{ra} \cup \{(v, d_v)\}$;

Batching in the algorithm is achieved by saving all the nodes identified as seeds in a special set B (line 13). This is referred to as "postboning" the seeds. Only after all the other open-vertices from set O have been expanded (meaning that the set O is empty after removing the vertex in line 10) the set B is used for loading all the circular ranges at once. Therefore, in line 25 a set of circles is loaded at once instead of single circles in MineR(X).

The disadvantages of the MineRB approach are the same as the ones for MineR(X) with the added problem of not being able to expire vertices from main memory. The performed queries are only doable by a spatial database system and might even a bit more complex when compared to the (also complex) queries used by MineR(X). The complexity of the algorithm is the same as for MDijkstra, Mine(X) and MineR(X).

5.3. Range Shape Variation

A way to deal with the disadvantages of MineR(X) is to vary the shape of the ranges loaded. Since multiple approaches within spatial data processing use bounding boxes, a concept using rectangular or even quadratic areas, using them for loading information during network expansion seemed promising. Spatial databases use these bounding boxes to index geometric and/or geographical data, so that containment of a vertex within a rectangular box is much faster than checking for inclusion in a circle. Another optimization, when using different shapes for range loading, can be performed by preventing overlapped regions. An approach to align rectangles on a map without overlaps is the usage of so called tiles. An algorithm loading data from the underlying data source with the help of tiles has been presented by Krismer et al. at the 29th International Conference on Scientific and Statistical Database Management (SSDBM) in Chicago, Illinois, USA in 2017. The following is taken from this very publication presenting the algorithm called "Multimodal Incremental Network Expansion using Tile regions (and vertex eXpiration)" or MineT(X) [65].

Tiles are well-known in the context of geoinformatics and are utilized by the widely used Web Map Tile Service (WMTS) Implementation Standard that has been defined by the Open Geospatial Consortium (OGC) [74]. Since the well-known term "tile" only refers to a "rectangular pictorial representation of geographic data", meaning the visualization of the data, but not the data itself, it is necessary to distinguish between tiles and tile regions. Hence, the term "tile region" is introduced, which also refers to the geographic data that lies inside a tile.

Definition 22: Tile region

A tile region is of quadratic shape and covers a part of the Multimodal Spatial Network. The size of a tile region is determined by a zoom level z.

To allow performance tuning, different tile region sizes can be implemented. This concept is also used in interactive online maps, but with one major difference. In tiles used by interactive maps, information on zoom level $l-1$ is aggregated from level l. In contrast, tile regions always represent the same data, only the size of the represented region changes. Zoom level z=0 refers to the maximal extent of the network (for geographic data this equals the whole world), while an increment to the level is equal to dividing each tile region of the previous level into four new ones. This is equal to a quad-tree partitioning of the network and means that zoom level z=12 already produces $4^{12} = 16.777.216$ different tile regions. The collection of all tile regions of a specific z form a matrix that is defined as tile region matrix.

Definition 23: Tile region matrix

A tile region matrix is the set of all tile regions for a fixed scale defined by zoom level z. The number of tile regions in a tile region matrix is determined by the following formula: 4^z

The collection of tile region matrices from level 0 (equaling a single tile region covering the whole world) to a certain zoom level z is also called a tile pyramid. This scheme has been described in 2012 by Garcia et al. [35]. Their visualization of a tile pyramid from level 0 to a specific zoom level (called l by Garcia et al.) is listed in Figure 5.2.

When loading ranges with tile shape, the first step is to determine to which tile region the vertex triggering the loading process belongs to. This is done by utilizing the PostgreSQL database functions “lon2tile” and “lat2tile” defined by the OpenStreetMap (OSM) community [94]. Secondly, the extent of the tile is determined by using the database functions “tile2lon” and “tile2lat” (again from OSM). The results are passed to the PostGIS function “ST_MakeEnvelope” in order to create an envelope used for vertex loading. The next step is to determine all vertices contained with this envelope by using PostGISs containment operator “ ” (depending on the database implementation, for some databases also the intersection operator “&&” is applicable and can be used interchangeably to the containment operator). Finally, all the information connected to at least one of the vertices identified in the previous

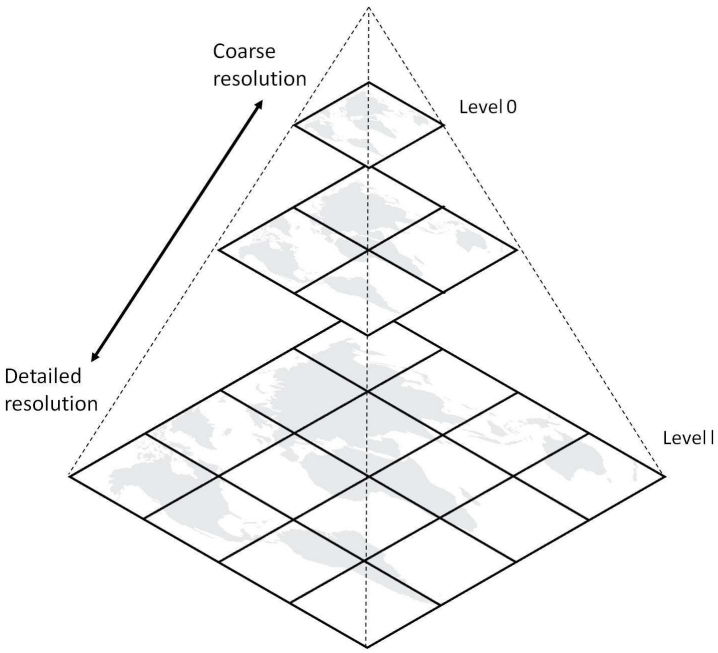


Figure 5.2.: Tile pyramid from level 0 to l.

step are loaded. Algorithm 6 lists MineTX in pseudo code, while changes between MineX and MineTX are highlighted in blue. The algorithm introduces the need for an additional parameter z defining the zoom level.

By using a tile region matrix with incremented z, the loaded chunks become smaller and the amount of data loaded within one request decreases. Therefore, a trade-off has to be made between the number of database requests and the amount of vertices held in main memory. The advantage of this strategy is that the algorithm can be easily tailored to the available hardware. If the database connection comes with high latency and if there is enough main memory, z can be chosen to be small (loading more data with one request). If main memory is small, z can be increased in order to load smaller tile regions. This approach also diminishes the problem that the amount of vertices held by one tile region varies within a tile region matrix.

Algorithm 6: Algorithm MineTX($q, d_{max}, s, t, z, \mathsf{N}$)

input : $q, d_{max}, s, t, z, \mathsf{N}$
output : E^{ra}, V^{ra}

1 $C \leftarrow \emptyset$;
2 $O \leftarrow \emptyset$;
3 $T \leftarrow \emptyset$;

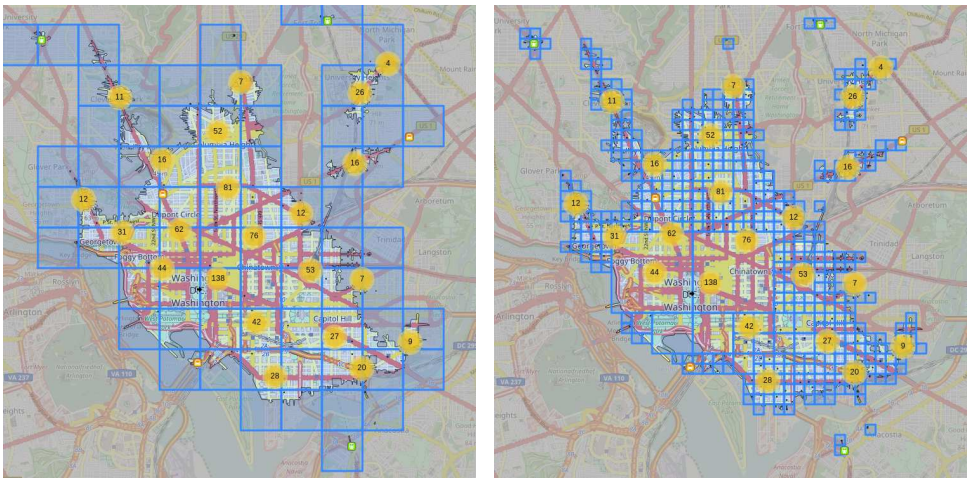
// project query point to start location (if necessary)
4 **if** q *coincides with* v **then**
5 $\quad O \leftarrow \{(v, 0, cnt_v)\}$; // update the duration of v in O to 0
6 **else**
7 $\quad P = \{(v,u) \mid e \in E : d_e = \min_{\forall x \in E} dist(q,x)\}$; // find edge(s) with min distance
8 $\quad$ **foreach** $e = (v,u) \in P$ **do**
9 $\quad\quad O \leftarrow \{(u, o/s, cnt_u)\}$;
$\quad\quad$ // add segments from q to edge (u,v) and to start vertex on edge
10 $\quad\quad E^{ra} \leftarrow E^{ra} \cup \{((q,o), 0, \lambda(q,o)), ((u,v), \max(0, ((o/s) - d_{max})), o)\}$;

// network expansion
11 **while** $O \neq \emptyset \wedge (v, d_v, cnt_v) \leftarrow dequeue(O) \wedge d_v \leq d_{max}$ **do**
12 $\quad O \leftarrow O \setminus \{v\}$;
13 $\quad C \leftarrow C \cup \{v\}$;
$\quad$ // load vertices by using a tile range
14 $\quad$ **if** $degree(v) = 0$ **then** // v is seed
$\quad\quad$ // v has not been loaded...
$\quad\quad$ // ...load all edges within the tile region defined by v and z
15 $\quad\quad T \leftarrow \text{loadTileRegion}(\{v\}, \{z\})$;
16 $\quad\quad$ **foreach** $(u', v') \in T$ **do**
17 $\quad\quad\quad$ **if** $u' \notin \{O \cup C\}$ **then**
18 $\quad\quad\quad\quad O \leftarrow \cup\{(u', \infty, cnt_{u'})\}$;
19 $\quad\quad\quad$ **if** $v' \notin \{O \cup C\}$ **then**
20 $\quad\quad\quad\quad O \leftarrow \cup\{(v', \infty, cnt_{v'})\}$;

$\quad$ // continue expansion like MineX would (starting from line 12)
21 $\quad$ **foreach** $e = (u,v) \in E$ **do**
22 $\quad\quad dTemp \leftarrow d_v + \tau(e, t + d_v)$;
23 $\quad\quad$ **if** $u \notin C$ **then**
24 $\quad\quad\quad O \leftarrow (O \setminus \{(u, d_u)\}) \cup \{(u, \min(d_u, dTemp))\}$;
25 $\quad\quad cnt \leftarrow cnt_u - 1$;
26 $\quad\quad$ **if** $u \in C \wedge cnt = 0$ **then**
27 $\quad\quad\quad C \leftarrow C \setminus \{u\}$;
28 $\quad\quad$ **else**
29 $\quad\quad\quad O \leftarrow (O \setminus \{u, d_u, cnt_u\}) \cup \{u, d_u, cnt\}$;
30 $\quad\quad$ **if** $\mu(\theta(e)) \in \{$"csct", "csdt"$\}$ **then**
31 $\quad\quad\quad$ **if** $dTemp \leq d_{max}$ **then**
32 $\quad\quad\quad\quad E^{ra} \leftarrow E^{ra} \cup \{(e, 0, \lambda(e))\}$;
33 $\quad\quad\quad$ **else**
34 $\quad\quad\quad\quad E^{ra} \leftarrow E^{ra} \cup \{(e, o, \lambda(e))\}$, where $d((e,o), q, t) = d_{max}$;
35 $\quad cnt \leftarrow cnt_v - 1$;
36 $\quad$ **if** $v \in C \wedge cnt = 0$ **then**
37 $\quad\quad C \leftarrow C \setminus \{v\}$;
38 $\quad$ **else**
39 $\quad\quad O \leftarrow (O \setminus \{v, d_v, cnt_v\}) \cup \{v, d_v, cnt\}$;
40 $\quad V^{ra} \leftarrow V^{ra} \cup \{(v, d_v)\}$;

Adaptive Zooming

After its publication at the SSDBM conference 2017 in Chicago, Illinois [65], slight improvements have been made to the algorithm. The most important one is that the algorithm can be configured to automatically adapt the zoom level z by the remaining travel duration. That way less information known as false positive are likely to be loaded and the parameter z does not need to be passed to the algorithm anymore. In Figure 5.3(a) a non-adaptive computation is shown, that always uses a zoom level of 15, showing the loaded tiles in a bluish color framed by a blue border. The used query point is located in the city center of Washington, D.C., while a travel duration of fifteen minutes is used. Figure 5.3(b) shows the exact same query, but with the usage of adaptive zoom levels. In the latter image the zoom level is 12 if the remaining travel duration is larger than 2800 seconds, changes z to 13 for a duration between 2001 and 2800, then increases z to 14 until duration is below 1800, using 15 as a zoom level between 1301 and 1800 seconds. For a duration between 801 and 1300 seconds a zoom level of 16 is used. Finally the adaptive case uses small tiles with a zoom level of 17 for queries if the remaining duration is below 800 seconds.



(a) Non-adaptive loading of tile ranges (b) Adaptive loading of tile ranges

Figure 5.3.: MineT(X) non-adaptive tile range size compared to usage of adaptive zoom levels in the city of Washington, D.C.

As can be seen, the size of the loaded tiles decreases as soon as the reachable area is further away from the query point, which is located at the Zero Milestone south of the White House (or in terms of the Figure 5.3 at the lower left quarter of the image). Although the adaptive approach increases the number of calls to the data source a bit, the data transferred is reduced and especially the number of vertices loaded as false positives.

The ranges used for the adaptive levels have to be determined somehow. There are multiple ways to do this. One is similar to the approach that is used during precomputation when using circular ranges in the MineR(X) algorithm. The number density of a vertex can be used in order to estimate the zoom level z when a certain remaining duration triggers the loading of the next tile range. The drawback of this approach is, that it needs an additional data structure or database table that records this information for every vertex for multiple time durations, but is able to deliver accurate tile sizes, such that for every tile loaded the amount of data transferred stays almost constant. Another approach, that has been used to determine the tile ranges used in the sample above, is to use the tested overall computation runtime as factor to optimize. Several computations were performed with random query points all over the dataset with varying travel duration and changing zoom level z. During these tests the computation runtime was recorded, so that for every random query point the result is a graph that records the time needed to calculate the area reachable within x minutes when using different zoom levels. In the end all the runtimes are merged together by simply adding the times recorded at a certain z with a fixed d_{max}. The result of this test is a figure plotting one function per tested zoom level z. With these information it is straight-forward to determine the best adaptive ranges. Starting with the smallest d_{max} recorded the zoom level delivering the best performance is noted. On every line intersection the new best z is noted together with the $dmax$ of the (the value on the x-axis) intersection point. If this is done for all the values then the result looks like "$800 : 16, 1300 : 15, 1800 : 14, 2300 : 13, 2800 : 12$" which represents the best adaptive zoom level ranges for the dataset under test. This procedure is also carried out during the evaluation in Section 9.5. A minor improvement to this technique has been applied as well. It makes sense to never increase the zoom level, since this would mean that for larger travel times smaller tile ranges are loaded. Such a behavior can only be caused by the randomness of the query points, but since longer travel times will always result in a bigger reachable area when starting at the same query point, the tile sizes loaded should at least stay the same or become smaller with decreasing remaining travel duration. If multiple different zoom levels deliver the same performance, a decision has been made to either load the biggest tile size (equal to the smallest zoom level), resulting in less data source accesses, or to load the biggest zoom level (equal to the smallest tile size). In the sample above the bigger zoom level has been preferred, to reduce the number of vertices loaded as false positives.

The complexity of the MineT(X) algorithm is the same as for MineR(X), and also as for MineRB, Mine(X) and MDijkstra. However, MineT(X) addresses some of the disadvantages of MineR(X). Since the tile regions are in fixed positions, the assignment of vertices to specific tiles (depending on a certain zoom level) can easily be precomputed. As a result, the requirement for the

need of a spatial database does not hold true. In theory, it is possible with the help of an additional database table (e.g. containing the columns "zoomLevel", "vertex" and "tileNumber") to replace the containment operator "~" with a simple join. However, a spatial database reduces storage needed and therefore MineT(X) combines the best of both worlds. A spatial database is not a hard requirement for the algorithm to work anymore, but it can benefit from its features in order to reduce the number of tables (and bytes) stored. The queries sent to the database are much simpler when compared to the ones of MineR(X). No circle has to be created and the containment within a tile region representing a rectangular shape, can directly be computed using index operations. No additional calculations have to be performed that check for inclusion in a circular region. However, there remains a disadvantage that are not solved by using tiles, namely the problem of false positively loaded vertices that are never expired from memory. This issue is relieved by the introduction of adaptive zoom level loading, but more advanced techniques are needed to further reduce the memory consumption and computation time when using MineR(X) and MineT(X).

5.4. Incremental Calculation of Isochrones

A well-known technique to speed up computations, that has not been applied to the field of isochrone calculation yet, is memoization. The term was coined by D. Michie in 1968 [78]. The idea behind memoization is to store the results of expensive calculations in a cache and re-use those cached results instead of re-computing them each time. This works for all deterministic functions whose results do not change throughout various calls. Therefore, this technique can also be applied to the computation of isochrones if enough information is stored in the cache. When computing isodistances, the query point together with a limiting distance is enough information for memoization. For unimodal isochrones additional information regarding date and time, maximal duration, as well as traveling speed has to be cached. Isochrones in Multimodal Spatial Networks need even more information, for example the traveling direction.

Memoization is often applied to reduce the computation time not only of already computed results, but also for iterative or recursive algorithms. One well-known example where this approach reduces computation time by far is for computing fibonacci numbers. Storing the last two results in a cache allows to compute the next fibonacci value with a single operation (by adding the two values from cache), instead of computing the whole fibonacci series. Based on this observation, the idea came into mind, if something similar could be done for isochrones as well. A reachable area created with a travel duration of

ten minutes will always include the area that is reachable within five minutes. However, one major drawback of the algorithms discussed in Chapter 4 is that they always compute the results from scratch without the knowledge of former computations. Since it is not trivial to implement an incremental calculation that is based on known results, various different cases that need to be taken care of, have been identified and were investigated in detail. A paper describing incremental calculation of isochrones regarding travel duration has been published by Krismer et al. at the "Grundlagen von Datenbanken" workshop (GvDB) in Bolzano, Italy in 2014 [67]. The following paragraphs are taken from this very publication and have been adapted to include research that took place since the publication.

5.4.1. Challenges for Incremental Calculation

There are some challenges that need to be addressed when implementing an incremental calculation. While for networks containing only continuous time edges, it is possible to expand or shrink the reachable area solely from its border, this is not true for discrete time edges. For those, new connections can become available, even if they are not located at (or near) the border, since more schedules have to be considered. To take care of this problem, all vertices connected via discrete time edges are added to a list *l_hubs*. These vertices are the ones we referred to as network hubs. Besides the hub h itself, further information is stored in this list: the time t of arrival (or departure depending on the computation direction) at the vertex and the remaining duration d that can be used. With this information it is possible to continue computation from any hub with a modified traveling time for the algorithms. When it comes to vertex expiration, it is important to understand that the cached result does not equal the disjoint groups introduced to implement vertex expiration, namely O, C or X. To be precise, the cached information equals the output of the algorithms, which is stored in variables V^{ra} and E^{ra}.

The list *l_hubs* needs to be stored in addition to the isochrone's maximal traveling duration and the isochrone result itself, so that it can be used for incremental calculation. None of this information needs to be held in memory during computation of the base isochrone itself and is only touched on incremental calculation. Therefore, runtime and memory consumption of the isochrone algorithms will not be influenced much.

Other problems include modifications to the spatial network in combination with incremental isochrones. If there is some change applied to the underlying network, all the base isochrones can not be used for incremental calculation any more. It can not be guaranteed that the network modification does not

influence the base isochrone. Changes in the schedules of modalities (for example the public transportation systems) could cause problems as well, as they would also influence the base isochrone. As a result, incremental calculation only work in static networks. If a network changes all the cached results will have to be removed or at least checked for validation.

5.4.2. Types of Calculation

There are three different cases that have to be kept in mind when calculating an isochrone with traveling time *dmax* using a base isochrone with duration *dmax_base*.

Case $dmax = dmax\_base$

The first and most simple case, is the one where *dmax* is equal to *dmax_base*. In these cases it is obvious that the calculation result can be returned directly without any further modification.

Case $dmax < dmax\_base$

The second case is the one where *dmax* is less than *dmax_base*. In this situation all vertices can be iterated and checked for suitability. If the duration is less or equal to *dmax*, then the vertex also belongs to the new result, otherwise it does not. As an alternative, the reachable area can also be created by shrinking starting at the border. The network hubs do not need any special treatment, since no new areas can become part of the result if the available time decreased. The only necessary task is the recalculation of the duration from the query point to the vertex in the isochrone. The duration d from the query point q to a network vertex v is then equal to (assuming that the border point with the minimal duration to v is named bp):

$$d(q, v) = d(q, bp) - d(bp, v)$$

Case $dmax > dmax\_base$

The remaining case, where *dmax* is bigger than *dmax_base*, is more complex. It differs in the fact that new, possibly disconnected areas can become part of the result and therefore it is not sufficient to look at all the border points of the cached reachable area. The new areas are caused by discrete connections from network hubs. A real-world example is a train station where a train is leaving at time *t_train* due to its schedule and arriving at a remote station

at or before time *dmax* (any time later than *dmax_base* is feasible). The time *t_train* has to be later than the arrival time at the station (and after the isochrones starting time).

Since only network hubs can create new disconnected reachable areas, it is sufficient to grow the isochrone from its border and all the network hubs. Because all these hubs have been store in list *l_hubs*, only this very list and the points at the border have to be considered.

Table 5.1 summarizes the recently mentioned calculation types shortly.

Case	Instructions
$dmax < dmax\_base$	iterating vertices from base isochrone + check if travel time is <= *dmax* or shrink base isochrone from border
$dmax = dmax\_base$	return base isochrone
$dmax > dmax\_base$	extend base isochrone from border points and with list *l_hubs*

Table 5.1.: Incremental calculation instructions

CHAPTER 6

Enhancements on Isochrone Application

Improvements to the algorithms that are able to compute isochrones in Multimodal Spatial Networks have not only been applied in terms of performance, but also to extend isochrone application capabilities. In this chapter four enhancements are proposed:

- *Averaged isochrones*: A single isochrone represents the reachability of or from a certain place at a given time. Averaged isochrones focus on answering the question on how good a place is reachable over a certain time span.

- *Time-invariant isochrones*: Time-invariant isochrones deal with the reachability within the time span regarded by averaged isochrones. If a place is reachable by bus quite well, e.g. 10 times within one hour, but all the buses arrive or leave within the first ten minutes of the hour, time-invariant isochrones are able to unveil that the reachability is not well distributed.

- *Elevation aware computation*: Computations with road networks are carried out without the knowledge of elevation. Throughout the past years most routing engines and applications have considered this issue, so that for travelers like cyclists routes are suggested that prevent steep hills and prefer declines. Google introduced this concept in its product "Maps" and its routing engine in 2014. The same concept is ported to isochrones, since data delivering information about elevation is now freely available for most regions of the world.

- *User tailored isochrones*: The reachability by bike is not only depending on the elevation profile of a route, but also on the training level of the cyclist itself. Therefore, the concept of user tailored isochrones is introduced. With the knowledge about the fitness of a cyclist, that for example can be gathered from previously recorded data, it is possible to tailor the computation to individuals. As a result a (semi-)professional cyclist will be able to reach a bigger area in the same time when compared to a untrained person. User tailored isochrones do not only change the speed of a cyclist, but also take personal preference into account. A fearless teenager might prefer to take a fast and even risky downhill track, while an untrained cyclist might take a detour in order to have a more flat route.

All these enhancements are implemented and can be used in real-world datasets. This is possible by using an application which has been developed throughout this PhD thesis and that is described in detail in Chapter 8 or directly by exploring the sources [61].

6.1. Averaged Isochrones

While for individuals traveling through a city the most important aspect is what can be reached when starting at a certain time, the same might not be true when planning in the long-term. If a new shop is to be opened or a house is to be built, it is of most interest how easy the building can be reached in general and not only when starting from a certain point in time. Consider a pharmacy for example. It is great if it can be reached by bus at eight o'clock a.m. and six o'clock p.m., but if in between it is only reachable by using a car, its location might not be optimal. The same holds true for real-estate, although the interval of the time points where it is easily reachable might differ. The availability of fast access to a train in the morning is great when traveling to work, but if there is no train for the children to return from school

or in the evening when returning from work, the benefit of the early morning train is diminished.

To take these considerations into account "Averaged isochrones" have been introduced. Such isochrones focuses on the reachability over a given time range. Depending on the use case the range may heavily vary. When opening a shop the reachability during the opening hours is most likely of interest, while for city planning the overall daytime could be important.

Averaged isochrones are easy to compute in networks without discrete edges. In this case the reachability over time stays constant. When traveling by bike and if a place is easily reachable during daytime, it will also be during night time. The same holds true for walking individuals or when traveling by car and there are no traffic jams. Traffic jams on the other hand would have to be implemented by continuous space and discrete time edges (or a similar approach), since due to its definition a continuous space and time edge is always passable. As a result, in a network with only continuous time edges an averaged isochrone equals any isochrone computed from the query point of interest.

However, if discrete time edges are part of the network, this assumption does not hold true. An approach to implement an averaged isochrone could be the adaption of the schedules utilized by discrete network edges. If schedules are not recording certain arrival or departure times, but by the frequency of availability within a time span (resulting in frequency based schedule), computations built upon such schedules will always result in averaged isochrones. The problem with this approach is, that the time span needs to be fixed. If the span determining the averaged isochrone is one hour, a frequency based schedule would have to be created using the exact same time span. For different spans, e.g. to twelve hours, the information recorded by the frequency based schedules would have to be re-created from the original schedule. Another disadvantage is that schedules would have to be kept multiple times. One schedule would be needed that uses arrival and departure times, another one with a time span of one hour and maybe even more depending on how many time spans are of interest. If only one frequency based schedule is used, while the original one is dropped from the data source, then it is only possible to compute averaged isochrones, but not those using a certain time and date instead of a time span. As a result, a different procedure was implemented to create averaged isochrones.

The approach does not focus on changing parts of the network or the schedules, but uses multiple equally distributed isochrones that are computed within the time span of interest. Staying with the example of a pharmacy, the time span

of interest was 8:00 a.m. to 6:00 p.m. (or 08:00 until 18:00). An averaged isochrone can also be created from twenty-two isochrones that are computed with an starting time varying by half an hour starting with 08:00 for the first and 18:00 for the last isochrone. This way, the accuracy of the resulting isochrone can be easily varied by changing the number of single isochrones used during computation. For simplicity Figure 6.1 shows a possible starting position for an averaged computation using only four single isochrones within a time span of one hour (so one isochrone starts at e.g 08:00, the next at 08:20, the third at 08:40 and the last at 09:00 o'clock). The query point here is the green point in the reachable areas. It location represents the entrance of the eastern graveyard in the city of Innsbruck, Tyrol, Austria.

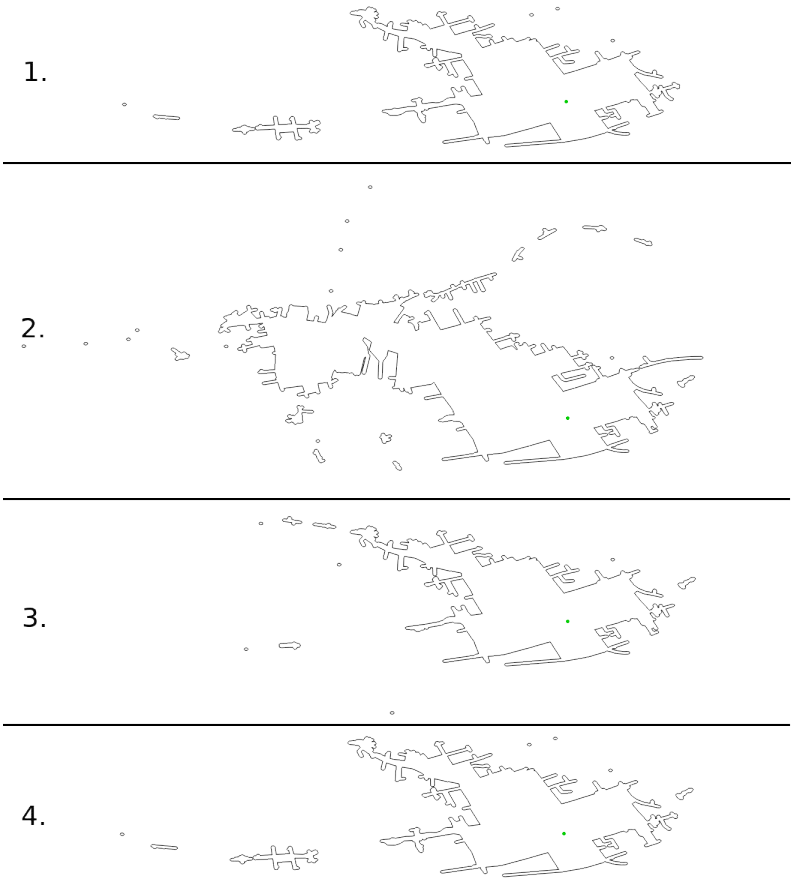


Figure 6.1.: Possible starting positions in Innsbruck for an averaged isochrone computation using four single isochrones.

These four isochrones are then processed in order to assign a frequency to every place reachable. This is done by utilizing the resulting geometries instead of the edges and vertices of the reachable area. Starting with the first two isochrones, the intersection between them is identified. The intersections are

then joined together into one (multi)polygon structure that is stored in a list at position two. The non-intersecting parts are also unioned, but are stored at position one. The positions in the created list structure represent in how many constructing isochrones the parts of the (multi)polygon were included. For the first step, two means that they are always reachable (100 percent for the two isochrones) and position one equals a reachability of 50 percent. Now the third isochrone is considered. It is intersected with the multipolygon at list index two, resulting in intersecting sections that are again joined to become a multipolygon that is stored at list position 3 (parts in that multipolygon were included in all creating isochrones). The non-intersecting parts are tested against the multipolygon from position one. Intersecting parts are promoted and form a multipolygon stored at list position two, while all the other sections become the new entry at list position one. Algorithm 7 lists the procedure in pseudo code, using an input array $I[]$ and a single polygon as output (A).

Algorithm 7: Algorithm AveragedIsochrone($I[]$)

input : $I[]$
output : A

```
// Initialize list L, skip entry 0 (leave it empty)
// (do not store polygon of area reachable by 0 percent)
1  L ← [empty, I[0]];
2  I ← I \ I[0];
3  foreach i ∈ I do
4      for j ← size(L); j > 0; j ← j − 1 do
5          temp ← intersection(i, L[j]);
6          i ← i − temp;
7          if j + 1 < size(L) then
8              L[j + 1] ← union(L[j + 1], temp);
9          else
10             L[size(L)] ← temp; // Append temp to list L
11     L[1] ← union(i, i);
// Return polygon from list with frequency just above fifty percent
12 A ← L[ceil(sizeof(L)/2)];
```

After all the constructing isochrones have been iterated, a structure is computed that gives information about the reachability of places in a percentage value. Entries at the first position are reachable by a percentage of $100/numberOfCreatingIsochrones$, while the places at the very end are always reachable (by 100%). The averaged isochrone then can be easily extracted by finding the correct position in the result list. For the definition of the averaged isochrone to be reachable by at least 50 percent, the position is the one right in the middle (length of the list divided by two; whereas real numbers are need to be ceiled). The result of the four sample isochrones from above looks like Figure 6.2(a).

The visualization of the result can be done with the help of transparent overlays. This adds some visual information as it not only shows the averaged isochrone, i.e the areas reachable by at least 50 percent of the time, but also assigns an optical reachability value for all the other places. In Figure 6.2(b) the transparency of the region coincides with the reachability. The transparency is computed by $1 - percentageReachable$. If the area is reachable within only one of the four isochrones used during computation, the transparency value is set to 25 percent, while places always reachable are shown as opaque (a transparency value of zero). During visualization this effect can be achieved by simply overlaying the transparent isochrones.

(a) Averaged isochrone (b) Visualization using transparency

Figure 6.2.: Averaged isochrone computation result.

The complexity of the algorithm itself is $\mathcal{O}(n^2)$, whereas the runtime heavily depends on the time needed for polygon intersection and union. Those two operations can be performed with a complexity of $\mathcal{O}(n * log(n))$ as shown by Lin et al. in [69].

6.2. Time-invariant Isochrones

The averaged isochrone considers a time span, but fails to take the distribution of the reachability into account. If schedules are not based on a fixed frequency that does not change for time segments within the same day, this can become a problem. In real-world scenarios this can occur due to rush hours. In the morning a city is most probably easily reachable, since school buses and a lot of trains are on duty, but the situation in the afternoon might be completely different. Therefore, the individual points in time a place can be reached, need to be analyzed. The result is called a time-invariant isochrone. An algorithm capable of doing this is based on the averaged isochrone computation, but is extended by additional steps. It is listed in Algorithm 8.

Algorithm 8: Algorithm TimeInvariantIsochrone($I[]$)

input : $I[], T[]$
output : $TI[]$

```
// Initialize list L
I ← ∅;
for i ← 0; i < size(I[]); i ← i + 1 do
    I[i] ← {(iso : I[i], timeMarks : {T[i]})};
L ← [I[0]];
foreach i ∈ I do
    for j ← size(L); j > 0; j ← j − 1 do
        temp ← (
            iso : intersection(i.iso, L[j].iso),
            timeMarks : L[j].timeMarks ∪ i.timeMarks
        );
        L[j].iso ← L[j].iso − temp.iso;
        L[size(L)] ← temp;
        temp ← (
            iso : difference(i.iso, L[j].iso),
            timeMarks : L[j].timeMarks
        );
        L[j].iso ← L[j].iso − temp.iso;
        L[size(L)] ← temp;
        temp ← (
            iso : i.iso,
            timeMarks : i.timeMarks
        );
        for k ← 0; k < size(L); k ← k + 1 do
            temp.iso ← temp.iso − difference(temp.iso, L[k].iso);
        if notempty(temp.iso) then
            L[size(L)] ← temp;
        L.splice(j, 1); // Remove old element from result list L
tDelta ← T[1] − T[0];
foreach l ∈ L do
    D ← [0]; // Create delta time array to compute skewness from timeMarks in
     l
    foreach t ∈ l.timeMarks do
        D[size(D)] ← (t − T[0]) + tDelta;
    D[size(D)] ← T[size(T) − 1] + tDelta;
    s ← 3 ∗ (D̄ − D̃)/σ_D; // Calculate skewness for area l
    TI[size(TI)] ← (iso : l.iso, skewnewss : s);
```

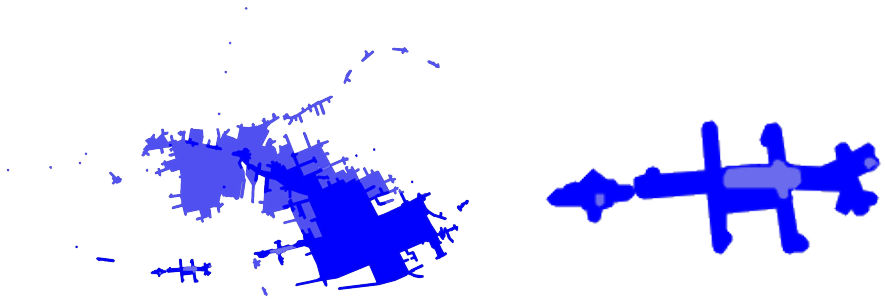
In difference to the averaged isochrone computation the algorithm uses geometric intersection and difference instead of intersection and union. This is necessary, since not only a percentage value of reachability is assigned to individual regions, but concrete time marks. Only by applying these marks it is possible to compute a skewness of the points in time a location is reachable. The determination of the areas to which time marks need to be assigned adds to the algorithms complexity, so that it is $\mathcal{O}(n^3)$.

After the algorithm finished its computation, an output array is available that contains multiple areas (defined by property "iso") with a skewness value assigned. The formula that is used to calculate the skewness of the time deltas in D has been defined by K. Pearson and therefore is known as Pearson's second coefficient (or median skewness) [24]. It is given in the following equation:

$$skewness = 3 * \frac{(mean(D) - median(D))}{standard\_deviation(D)} = 3 * \frac{(\overline{D} - \widetilde{D})}{\sigma_D}$$

There is one improvement over using the plain median skewness on the time marks. The time marks are extended with one additional mark at the beginning and one at the end, while shifting all the marks by the minimum time difference. There are two reasons for that. First, the skewness can also be computed when only one time mark is available. This would not be doable without this extension, since the skewness of a single value is not defined. The standard derivation of a single value is zero and since this deviation is needed as a divisor the difference of the mean and median of a distribution by its standard deviation a division by zero would occur. This would result in an infinite value. To overcome this issue, the skewness has been defined as undefined for a single value. Second, this extension enables to determine statistics on a more global level. In the case when an area is reachable for example the first two times, but not anytime later, the skewness would still be zero, since median and mean would be the same. With the help of the added values at the beginning and at the end the mean can only be the same as the median, if the time mark is exactly in the middle of all the occurring time points. The result is what would be considered as an equally distributed reachability over the whole considered time span, not only within the time marks of the currently iterated subpart of a single isochrone.

If these areas are visualized with the help of transparency, e.g. using the skewness as percentage of transparency with a minimal transparency of 25 percent, the output looks like Figure 6.3(a). For creation, the same isochrones were used as for the sample in Section 6.1 that are listed in Figure 6.1.



(a) Visualization (b) Zoomed detail

Figure 6.3.: Time-invariant isochrone.

The transparency in the result does not refer to a reachability, but to its distribution. Therefore, some effects can not be interpreted without the knowledge about the reachability of the place itself. As an example the area in Figure 6.3(a) at the bottom left can be seen, which is a so called reachability island caused by a bus connection. The opaque blue island surrounds three more transparent regions. This part of the time-invariant isochrone has been zoomed in Figure 6.3(b). The three transparent parts represent bus stations that are reachable three times out of the four creating areas (in isochrone #1, #3 and #4 for the two left bus stations and #1, #2 and #4 for the right bus station). The surrounding is reachable only twice, but with a better distribution (in isochrone #1 and #4 only). The skewness of the surrounding therefore is zero, while this is not true for the bus stations, where it is negative for the two left stations and positive for the right station.

However, in combination with the averaged isochrone the time-invariant isochrone adds valuable information to the computation result. With it statements can be made if a place is reachable during certain times only, e.g. during rush hours, or equally distributed throughout the whole time span. Since transparency can be used to visualize both results, it is possible to overlay them using multiple layers. Depending on the actual colors and the blending mode that is used to overlay the partially transparent layers, visualizations can be created that highlight places that are part of both, the averaged and time-invariant isochrone. These findings have been published at the AGIT poster session in 2015 by Krismer et al. [63].

6.3. Elevation Aware Isochrones

One thing that is missing for isochrones, that becomes obvious in the region of Tyrol, is elevation awareness. During computation no information about the elevation of a vertex, the steepness of a way or the gradient of the terrain, is used. To overcome this limitation, two things were performed. First, the information about the elevation was added in the datasets isochrones are computed in. Second, the information needs to be used by the algorithms. The latter is described in this section, while the first point is discussed on dataset creation, which is explained in Chapter 7 and Section 7.4.1. For the remainder of this section it is assumed that each vertex in the underlying network contains information about its location (e.g. by latitude and longitude) and about its elevation (given in meters above sea level). Furthermore, more vertices have been added to the networks graph, so that no inclines and declines are missed on long, straight roads.

A first examination when thinking about elevation is that it does not matter for the public transportation system. Here, the schedules applied to the discrete space and discrete time edges implicitly include the elevation. If a subway train takes five minutes from the main station in downtown to another station in uptown, elevation is not of interest. The reason is that the information used by the isochrones algorithms is always time. In schedules this is already included in the duration or the arrival/departure times, so elevation will not be of any use for schedules. This is completely different for continuous space edges. Here the information about the duration that is needed for traversal is given as length of the edge and not directly as time. In such cases the duration that is needed is computed from the length and a traveling speed on the edge. The speed that is reachable on a particular edge, however, is influenced by its slope. Therefore, regarding elevation will make a difference in the result for networks including such edges.

When thinking about roads in real-world networks, the achievable speeds are not only influenced by a slope, but also by the surface of a road. Even more important for elevation awareness is, that the change in speed depends on the surface and type of the road. If a cyclist drives down a steep decline the speed might increase fast if the street is paved or made out of asphalt. If in contrast the way is unpaved and made out of gravel, the increase in speed might be less dramatic. The same holds true for inclines and other means of transport. The type of the road itself also effects the obtainable speeds. On a cycle path, riding a bicycle requires less attention to the surrounding traffic as it does on a busy federal highway, so in addition to the roads surface also the type of it is of interest.

The transport mode and the surface on an edge are not the only things that influence the speed when regarding elevation. Another thing is the transportation modality itself. When driving in an overpowered sports car, an incline of twelve percent might not make any difference at all. The same steepness will be a problem for most cyclists and especially for non-athletic people carrying a heavy load up a mountain. The mode of movement might differ in the behavior for inclines and declines. Though cyclists will be faster riding downwards and slower going upwards, the effect of walking downwards will be less dramatic. For walking and cycling going upwards for long distances will slow down the average speed quite a bit, while in contrast walking down declines will not speed up walking as fast as it does for cycling. Hence, the effect is non-linear. A decline of thirty percent will increase traveling speed on a bicycle quite fast it will not increase the speed three times faster on a ninety percent decline.

All these observations indicate that different treatments for elevation information is needed for different modalities. Therefore, the following methods have been implemented (sorted by computational intensity):

- Schedule-based: as explained before, this is trivial, because schedules already implicitly include elevation data.

- Fixed-mode: uses a fixed speed that is applied for all edges in a network. This mode is for example used when computing isodistances.

- Simple mode: usable for modalities that are not influenced by elevation, such as cars. It models the behavior when not regarding elevation and can be used to compare results in three dimensional datasets (with elevation information available) with results from two dimensional ones (without elevation information). When using this model the resulting speed that an edge can be traversed with is solely defined by the maximum speed allowed and the maximum speed the vehicle (a car, a motorcycle or a moped...) is capable of.

- Cycling mode: a specialized mode for bicycles that uses specific average speeds per way type (e.g. 18km/h on a cycleway and 12km/h on an unpaved path). If further information about the surface of the way is available, it is used to adapt the speed by multiplying a so called "surface speed factor" (no change on an asphalt or unknown surface, reduced speed on muddy street by multiplying with a speed factor of 0.6). However, this mode does not take elevation information into account.

- Walking mode: a specialized mode for walking. Essentially implements the same methodology as the cycling mode does, but with speeds (and surface speed factors) suitable for walking.

- Elevation aware cycling mode: Extends the cycling mode with elevation awareness that uses a specific weighting function tailored to cyclists.

- Elevation aware walking mode: Adapts the walking mode and adds elevation awareness. Therefore, it can be used for hiking and walking. To calculate speed the rules from DIN 33466 are adapted. This standard has been introduced to estimate walking times and is used on hiking trail signposts in the middle of Europe.

The latter two modes are aware of elevation. To compute correct speed values in these modes several approaches have been researched. At first, a very simple method has been implemented that takes the start and end vertex of a networks edge and calculates the slope by simply applying the arctangent function. Since two edge lengths are known (the distance of the vertices and their delta in elevation), this gives the angle. For the elevation aware walking mode DIN 33466 is applied with a slight modification, if the angle is above a lower bound of two degrees. If the angle is below this threshold, elevation data will not be taken into account, since it can be assumed that walking (or cycling) on such slopes is not different than on completely flat ways. The formula used by this DIN is given in Algorithm 9:

Algorithm 9: Algorithm DIN 33466($v1, v2$)

input : $v1, v2$
output : $secTotal$

$timeFlat \leftarrow distanceInMeters(v1, v2)/4000$;
$deltaElevation \leftarrow max(v1.height, v2.height) - min(v1.height, v2.height)$;
$timeElevation \leftarrow 0$;
if $deltaElevation > 0$ **then**
  $timeElevation = deltaElevation/300$;
else
  $timeElevation = deltaElevation/500$;
$timeTotal \leftarrow min(timeElevation, timeFlat)/2 + max(timeElevation, timeFlat)$;
$secTotal \leftarrow timeTotal * 3600$;

As can be seen from these equations, DIN 33466 uses a time calculation based on two steps. The first step is to compute the time between the points as if there was no elevation (which is saved as time in hours in variable $timeFlat$). The rule uses a walking speed of 4000 meters per hour for this computation. The second step includes the delta in elevation and uses a speed of 300 meters

per hour for inclines or 500 meters per hour for declines. The result is then computed by adding the smaller value divided by two to the bigger value. This gives the time in hours, so that multiplying by 3600 is needed to get a value of seconds. For isochrone computation a small adaption is applied to the algorithm. Instead of 4000 meters per hour a base speed that has been calculated by the walking mode is used. Therefore, the elevation aware walking mode is a DIN 33466 enabled walking mode (as the name already suggests).

For the cycling mode no matching DIN or ISO norm could be found. Additional research has been carried out [70]. The results have been published at the GvDB workshop 2016 by Krismer et al. in Nörten-Hardenberg [66, 114]. Besides comparing and optimizing different digital elevation models, that will shortly be discussed in Chapter 7, a mathematical function suitable for time estimation for cyclists was found. For declines the resulting formula is

$$speed_{decline} = \sqrt{1 + 30 * slope_{decline}} * speed_{base}$$

The equation for inclines is

$$speed_{incline} = (1 - 5 * slope_{incline})^2 * speed_{base}$$

Together the two functions form a S-like curve, that increases the speed very quickly once a cyclist rides downhill and decreases the speed quickly once he rides uphill [70]. The base speed in those functions is determined by the cycling mode. As a result the elevation aware cycling mode is an extension of the cycling mode (as the name already suggests).

One of the problems that occurs with elevation data, is that taking the start and end vertex of an edge into account is not very accurate. The reason for this is, that the accuracy depends on the length of the edge. This approach is feasible for short way segments, but is problematic for longer ones that are used for example when modeling highways. In such cases there is the possibility that inclines and declines are lost. This can be seen on mountain streets for example. If there is a long straight way that is modeled solely by two vertices (start and end), it could happen that both vertices have a elevation of the same height, although in between of the two vertices there is a hill.

To overcome this disadvantage, two approaches seem feasible. The first one is to extend edges with intermediate vertices. If the resolution of the elevation model is large enough so that all elevations of interest are part of it, the number of locations necessary to regard all available information is determined by the

Nyquist frequency [112]. It has been defined for signal processing but is also applicable for this very case. If an elevation model has a resolution of 30 by 30 meters, then the frequency of the intermediate points needs to be at a maximum of 15 meters in order to reconstruct without losing information. This can be realized by segmenting roads and adding intermediate points before merging the elevation data. The drawback of this approach is that many intermediate points need to be regarded during elevation computation and that it adds a lot of points to the resulting network graph. This increases memory consumption and computation time of the algorithms that work on such a graph. Therefore, a second approach has been implemented that combines the best of both worlds. It regards elevation data beyond start and end vertex and does not include addition of intermediate points. In a separate preprocessing step information about the elevation profile of an edge itself can be computed. Only during this preprocessing step intermediate points are temporarily added to the edge. With them an average incline, average decline and the distance an edge inclines can be computed. The information can then be stored alongside the edge without the need to keep the intermediate points in the graph. With the help of the two average values elevation between the start and end vertex is not lost while keeping the computation itself fast.

6.4. User Tailored Isochrones

Another advantage of using a way type and a speed to compute the time that is needed to traverse an edge, is that it allows to tailor results to individual users. The following approach has been researched together with M. Malfertheiner [70].

Movement in general is a very individual task. Riding a bicycle can be seen as an example here. Driving steep and long slopes can be really tough for a non-experienced person. It is impossible to have a single setting that fits for all types of users. In general, an application should already suggest different paths for a mountain bike and a racing bike, but in the best case it tailors routes to individuals. A hobby cyclist might take two hours for a track, a professional racer takes 40 minutes and a person new to cycling takes three and a half hours. Therefore, it is necessary to find a mechanism, that is powerful enough to create a detailed profile of a user.

Such a mechanism can be found by assigning a collection of average speeds for various way types to every person. To regard elevation, the average speeds are splitted by slope. A way type is not assigned to a single speed value, but to 61 speed values (for slopes between -30 to +30 degrees). The resulting

matrix is quite big. The datasets described in Chapter 7 are best usable with 16 different way types, so the resulting matrix is of size 16x61. Every row of the matrix corresponds to a way type and every column to a slope value ranging from -30% to +30%. A cell is either null (no data inserted so far) or contains distance and speed. The schema of a profile is listed in Table 6.1, whereas in each cell the information about distance (d) and average speed (s) is stored. There can be a lot of empty cells in this table, so missing data must be handled somehow. For further classification such a user profile is referred to as "raw profile", while optimized profiles are called "user profiles".

		Slope		
		-30 % ... 0 ... + 30 %		
Way type	0 ... 15	[d,s] or *null*	...	
		...		

Table 6.1.: Raw profile matrix

The definition of the way types is done with the help of metadata that is assigned to each edge within the road network. The rules that are for this classification have been defined within [70] and are appended to this thesis. They are given in Table A.1.

It is obvious that a user can not manually enter his or her profile. This task would be much too complicated and error-prone. Therefore, a method is introduced to automatically generate user profiles from previously recorded tracks. Tracks can be recorded by any tracking device, be it a global navigation satellite system (GNSS)-enabled travel recorder, a navigation device or a GPS, GLONASS and/or Beidou enabled mobile phone. The only requirement is that tracks can be recorded and exported from the device. Such tracks are stored in the GPS exchange format (GPX). Some devices may need conversion from a vendor-specific format. GPX data looks similar to Listing 6.1 [70].

```
<trkpt lat="46.6502751969" lon="11.5866007190">
    <ele>543.28</ele>
    <time>2015-09-06T07:58:04Z</time>
</trkpt>
<trkpt lat="46.6489102878" lon="11.5852427669">
    <ele>542.32</ele>
    <time>2015-09-06T07:58:34Z</time>
</trkpt>
```

Listing 6.1: GPX file example

With the knowledge of time, elevation and position (given by latitude and longitude), it is possible to extract information needed to create a raw profile. The distance can be computed using the positions of two points assuming a straight line. Speed can be obtained using distance and time information. Finally, slope can be computed from the elevations at the positions. To improve the elevation values, it makes sense to apply a Kalman filter [54] to those values, if it has not been done by the recording device. To reduce noise in the signal even more, averaged values are taken every 200 meters [70]. The last thing that is missing, is information about the way type. To determine it, the track defined in the GPX file is snapped to the edges in the road network. From the matched edges metadata is extracted in order to determine the way type. This process is known as "map matching" and described in detail by M. Malfertheiner [70].

After a raw profile has been created with the help of tracks defined in GPX files, it needs to be further processed. The reason for this is, that it will most likely contain a lot of empty cells (null values). The fact that not all cells in a raw profile will be filled, leads to the definition of a profile as two dimensional matrix. One could even use a three dimensional model where the added dimension equals the ways surface. However, surfaces are either part of the way type definition or are rarely known during dataset creation. Therefore, for surfaces a correction factor (the surface speed factor) is used instead of tailoring the behavior on different surfaces to individuals.

Missing rows are not that much of a problem in a raw profile, e.g. people driving around a lot with cars do not need information on how fast they would be on a designated cycle path, since they are not allowed to drive those ways anyway. However, empty cells in a row constitute a problem. That means that speed values for e.g hiking paths are available, but not for all slopes. This is solved by curve-fitting which also helps to correct incorrect values of the raw profile that can occur if the tracking device has a too low sample frequency or delivers inaccurate positions. Since speeds should decrease on inclines and get faster on declines, a suitable mathematical function is an inverted Sigmoid-curve [45]. In order to fit this S–shaped curve to the samples from the raw profile, a special version of the Sigmoid curve, the so called inverse logistic function (or inverse logit function), is used [66]. It is extended by an addition of a minimal value d:

$$f(x) = d + L * (1 - \frac{1}{1 + e^{-k(x-x0)}})$$

- $d =$ the curves minimal value
- $L =$ the curves maximal value (the upper bound; largest value is $L + d$)

- k = the steepness of the curve (its "growth rate")
- $x0$ = the x-value of mid-point of the curve (its inflection point)

With the help of this function a noisy and incomplete raw profile can smoothed. The entries from the raw profile are weighted by the distance stored in it, so that the importance of more accurate values (that were collected from multiple way segments or longer ways) is increased. In Figure 6.4 a possible curve fitting result is shown for a cyclist on way type "$SMALL\_WAY\_PAVED$". The size of the points from the raw profile equals their importance and is used for weighting.

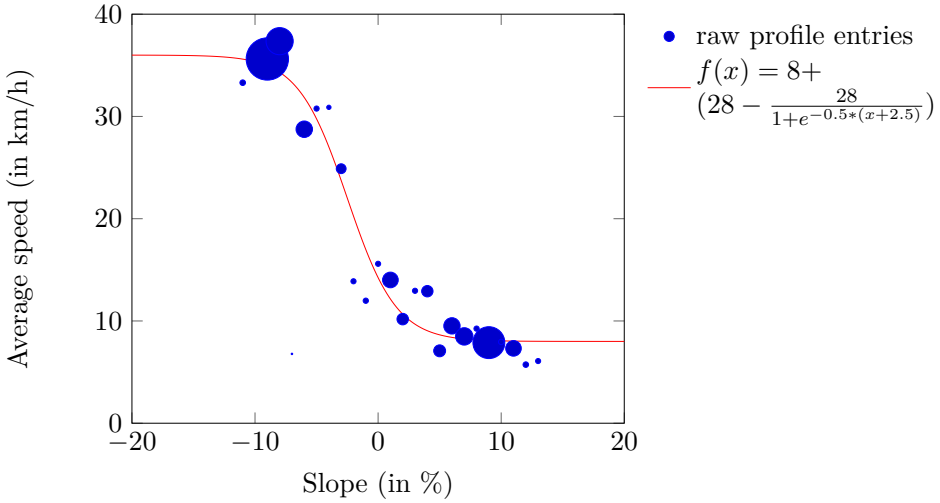


Figure 6.4.: Raw profile for a cyclist on a small paved way.

The Sigmoid curve that is used for smoothing will correct the outlier at slope -7% and will also fill empty values in the profile (the values from -30 up to -11 and from +13 until +30). This method of curve fitting only works if the data points are well distributed. To overcome this problem, only way types with entries summing up to a total distance of at least ten kilometers are used. In addition, artificial points (known as control points) with a small distance of 50 meters are added from the speed methods described in the previous Section 6.3. For way types missing in the raw profile or for those not matching the minimal distance criterion, speed computation falls back to using non–user–tailored methods which have been described in Section 6.3.

The resulting profile, the so called "user profile" can then be used during isochrone computation to look up speed values on specific way types and slopes and therefore is able to tailor the computation to individuals. Therefore, the performance of the isochrone algorithms is not diminished when using the smoothed user profiles.

CHAPTER 7

Generalizing Dataset Creation

To compute isochrones in Multimodal Spatial Networks, data about these networks needs to be available. Besides synthetically generated networks that are created with the help of mathematically modeled rules, datasets representing networks that model regions of the real world are of most interest. A first real-world dataset representing a Multimodal Spatial Network has already been shown in Figure 3.11.

A major problem when using transportation networks including schedule based means of transportation, is that there is no standard or even a best practice on how such datasets should be generated. Even more problematic is the fact, that there exist no freely available Multimodal Spatial Networks that can be used for comparing own research with existing approaches.

Therefore, a workflow is defined throughout this chapter during which a dataset is created. The whole process is an implementation of the extract and transform parts of an Extract, Transform, Load-process (ETL-process) that is well known from the field of Data Warehouses. The described dataset creation workflow has been published by Krismer et al. at the AGIT conference 2016 [64]. An implementation of this workflow is freely available as open-source [61], which has been termed “OpenStreetMap and Public Trans-

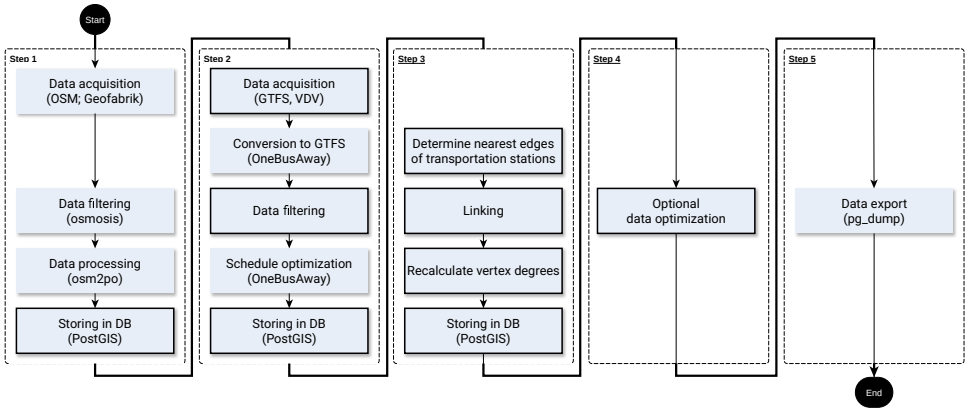


Figure 7.1.: Dataset creation workflow.

port Information to Multimodal DataSets" (osmPti2mmds). The steps that are needed to create datasets are shown in detail in Figure 7.1.

In general, there are five major steps, whereby each one will be described in a separate section:

- *Step 1 - Street network extraction*: Discusses the selection of appropriate data source from which information about road networks can be acquired. Furthermore, data filtering and optimization of the resulting graph is done in this step. After all actions have been carried out, a network represented by solely continuous space and continuous time edges is available.

- *Step 2 - Data retrieval and format conversion for public transportation systems*: Explains how data about schedules for public transportation systems is converted, filtered and optimized in order to be suitable for isochrone computation.

- *Step 3 - Linking different data sources*: In this step a method is established, that is used to link the road network (created in step 1) with the public transportation system data (from step 2). After this step the underlying graph represents a Multimodal Spatial Network (MMSN).

- *Step 4 - Optimizing and extending data*: In this step information from additional data sources, is combined with the MMSN. Although isochrones can be computed in these networks when skipping this step, the performed actions are a mandatory step for multiple algorithm enhancements from Chapter 6. The preprocessing steps for the MineR(X) algo-

rithms, as well as the elevation data preparation is carried out in this phase.

- *Step 5 - Data export*: The last step in the workflow is to export the prepared dataset. This is done to enable comparisons between researchers, to ease evaluation across various setups and to respect the license of various data sources used during dataset creation. Exports can be converted into various database formats, e.g. database dumps PostgreSQL/PostGIS, Oracle Spatial, Sqlite/SpatiaLite and Neo4J Spatial.

The process of dataset creation implemented by osmPti2mmds allows to create time-dependent network models. To some extend also the creation of time-expanded models is available in osmPti2mmds. However, since optimization and data extension has only been implemented for a time-dependent model, for the remainder of this chapter such a time-dependent model is assumed. It can also be noted that datasets are always created from scratch. Although an incremental method could be implemented for the road network and for transit schedules, the proposed approach always computes from scratch. That is because updates within the graph would affect running computations and that actions carried out during dataset optimization would also have to support an incremental approach, which is not trivial especially for the vertex density computation.

7.1. Street Network Extraction

It is obvious that when computing isodistances and isochrones in transportation networks, information about streets is mandatory. However, information about them can not be easily acquired, since in many parts of the world such data is not freely available. Well-known services from Apple, Google, Here and Microsoft are offered under a proprietary license. As a result, they can be accessed using an Advanced Programming Interface (API), but do not allow the export of the underlying data. Therefore, these providers are not feasible for the creation of a graph and the development of algorithms. For some countries, data can be obtained from the government services directly. This is for example true for the United States, where the Topologically Integrated Geographic Encoding and Referencing system (TIGER) is licensed under public domain, and for Austria (since 2016), where the formerly non-free dataset "Graphintegrations-Plattform" (GIP) has been licensed under Creative Commons 3.0 AT. There still remains the problem that data is not freely available for other states of the world. Even for publicly available information, different

data formats are used, making the data hard to compare and research on them cumbersome.

These circumstances led to the development of the OpenStreetMap (OSM) community project that was founded in 2004 by Steve Coast. It aims to support and enable the development of freely-reusable geospatial data, including information about streets. Work about OSM quality has been listed by Krismer et al. [65]: "Quality analyses have been a topic of research since the beginnings of the project [56]. Depending on the geographic region it varies [38, 80], but in contrast to other datasets it is of near global coverage, up-to-date, accessible for everyone without paying fees. The quality of the road network, which is of great importance when computing isochrones, has been examined by various researchers and was summarized by Neis et al. in [86] and more recently by Brovelli et al. in [15]. Especially in urban areas data is of good quality, whereas it can be poor in rural areas."

At the time of writing, OSM includes a little more than 110.500.000 ways all over the world [121]. According to M. Maron, who measured completeness of the road network by comparing its length against the CIA Factbook [30] in 2015, roads are well covered in OSM for all continents. The research performed by Maron led to the development of an application [71] that allows to carry out this completeness check for different regions of the world. There are still states left with a low road network coverage, especially China and India. In addition, comparing only the length of the road network leaves a hint about the coverage, but does not include an evaluation about the metadata of streets, e.g. the maximum speed allowed on a highway. However, many of the freely available datasets have been integrated into OSM, making it the best choice for a real-world road network source.

OSM is not only suitable because of its data quality and completeness. It allows to export the road network or parts of it at the database level and does not only offer services to access the data. Geofabrik [36] for example offers downloads of OSM database exports that have been sliced by geographical regions, mostly states. Slicing regions out of the so called "planet file" is a common strategy to ease the handling of geographical data collections. The elaborated dataset creation workflow uses the same mechanism in order to create datasets of city and state size. During this process, so called "boundary polygons" are used for data filtering. They are created from OSM data itself, and besides dataset creation they are also used in the application presented in Chapter 8.

For the application of isochrone computation it is further important to introduce the concept of routable graphs. What can be seen from the visualization

and especially from the placing of the vertices, is that there are a lot of unnecessary ones when it comes to routing. Vertices are often used to define the shape of a street within a road network, which can be seen on curves of streets in Figures 3.7 and 3.11. Since no left or right turns or junctions are involved on these vertices, they can be omitted when it comes to routing. The resulting routable graph is shown in Figure 7.2.

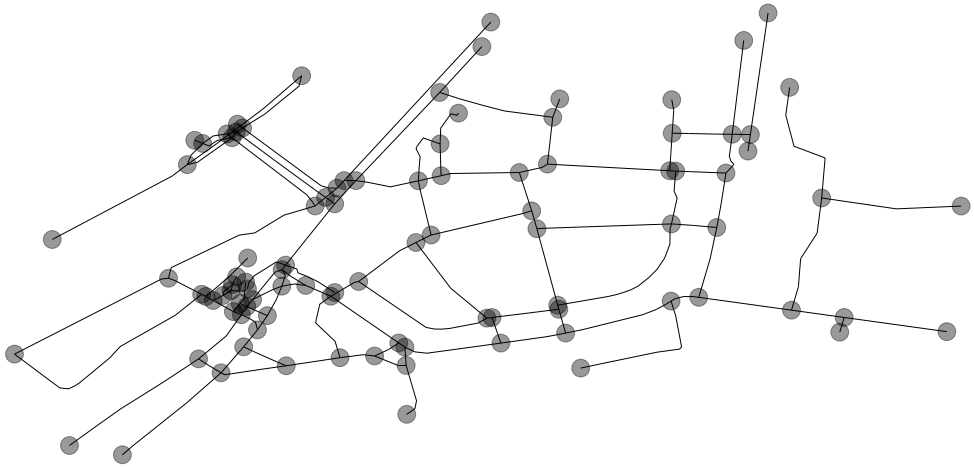


Figure 7.2.: Spatial routable road network modeling the city center of Innsbruck.

To implement a routable graph, a hypertournament can be used. The geometry of streets is then modeled with the help of hypervertices, while only the starting and ending vertices need to be regarded by the routing and isochrone algorithms. Depending on the data this reduces the number of vertices by far (in Figure 7.2 a little more than 51% of vertices can be omitted without loosing routing information). Only when regarding the length of the edges the very shape needs to be considered, so that when converting a graph into a routable graph the information about length is stored as property aside the others edge attributes. When it comes to OSM, attributes commonly stored with edges representing streets are maximum speed allowed, surface, way type, road width, number of lanes, a oneway flag as well as information about vehicles allowed to use the street. Therefore, all the information needed during isochrone calculation is at hand.

The creation of a routable graph out of OSM data is done with the help of osm2po [79]. This software allows for sophisticated configuration of the data processing step. It also is extensible so that all the information from OSM, such as way type and surface, can be included. The result of the data processing are SQL-files that then can be used to import the routable graph into a database. The format uses a relational scheme, that stores directed edges identified by start and end vertex. Information about the intermediate

vertices on such an edge is stored in a separate geometry column with a specific datatype that supports the "Simple Features" specification defined by the OpenGIS Consortium (OGC) [91]. If more information about the vertices are of interest, they need to be joined with a separate relation, that can also be generated from the created SQL-files.

7.2. Public Transport Schedules

To include information about public transportation and the schedules assigned to such systems, data has to be gathered from multiple sources. OpenStreetMap does not provide or even seeks to provide schedules for the public transportation system. The reason for that are besides copyright issues, that schedules change quite often and need regular updates. If they are not provided by the defining companies themselves, they are hard to keep up-to-date. The founder of the OSM project, Steve Coast, tried to implement a solution which was called "Transiki" [95] in 2010 anyway, and failed only half a year later.

The General Transit Feed Specification (GTFS) has become a de-facto standard for public transportation schedules. As a result, a growing number of transportation companies are providing files in this format. These files can be used to enrich the graph created in the previous step with discrete space and discrete time edges, making it a truly Multimodal Spatial Network. There are also some approaches like Transit.land [72] and Transitfeeds.com [20] that collect GTFS files from various public transportation companies, easing to find regions for which a Multimodal Spatial Network exists.

For some cities information is available in other digital formats, including HAFAS, DIVA, VDV452 or TransXChange. In such cases a transformation step is necessary that converts the data into the GTFS format for the workflow to handle it. However, this conversion introduces a potentially erroneous step and should be avoided whenever possible.

As soon as data is available in GTFS format, a filtering on the data and some optimization is performed. The filtering process basically targets the same as the data filtering does for geographical data in the first step. The schedules are narrowed down to a specific bounding polygon, in order to reduce the amount of data that needs to be stored in the database. The optimization step restructures the information again to aim for less data without losing relevant information. This is possible since GTFS allows to store the dates on which a certain bus is available in multiple ways, either as single date

values or as re-occurring weekdays within a time span. Besides the schedules a GTFS file also contains information about the location of stations as well as trips connecting the stations. Therefore, a graph that is enriched by schedule information can be created from these files.

At the end of this step, the data about the road network and the schedules of the public transportation system can be stored in a database side-by-side. This allows merging both information in the next step.

7.3. Data Merging

Although all the information needed about the road network and the public transportation systems is stored aside in the database, it is not connected. This means that the network will not allow for a multimodal usage and stick to the starting modality throughout computation. This is changed in step three of Figure 7.1, the data merging step, that interweaves the road network with (possibly multiple) public transportation schedules. In order to connect the various networks, the vertices of the underlying graphs need to be connected somehow. This is done in a three-step-process that is carried out for every vertex in every public transportation system:

- *Vertex projection*
- *Linking vertex creation*
- *Weaving*

At the beginning the vertex from the discrete public transportation system graph is projected to the continuous road network. Starting from the discrete vertex, the nearest edges in the continuous network are determined. Then, the nearest position on the edges to the discrete vertex is determined. It either equals the start or end vertex of the edge or is located in between. When the location equals an already existing vertex, the second step can be omitted, continuing with the very vertex as the linking vertex. If the location is located on the edge between start and end vertex, the found edge(s) is/are connected by using an orthogonal line on the edge through the discrete vertex. In difference of just using the nearest vertices from the continuous graph, this method guarantees to use short graph interconnections. Basically this approach equals the method that is applied when projecting a query point into the road network at the beginning of the algorithms described in Chapter 4 and 5.
At the intersection between the orthogonal line(s) and the edge(s) in the continuous graph, newly created vertices are added to the continuous graph, if there were no vertices at that position in the graph.

From this vertex two edges are created between the new continuous vertex and the discrete vertex connecting the various modalities (one for each direction). These edges are called graph interconnection, graph links or simply links throughout this thesis. The described approach is shown in Figure 7.3.

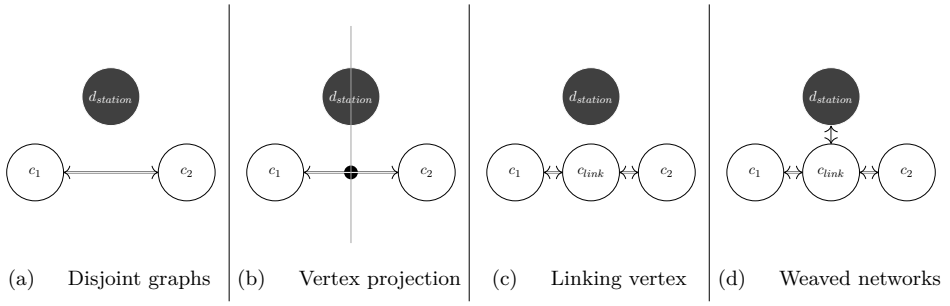


Figure 7.3.: Network graph weaving.

After the various networks have been weaved the different network vertices are iterated to check for identical geographical position. If multiple vertices were found at the very same location, then graph links are inserted between all of them. This allows to change modalities across various public transportation systems that have not been part of the same GTFS file.

Since vertices have been deleted and added, links could have been added to existing vertices. The information about the vertex degree is stored by osm2po in the database, but after the data merging the values are incorrect. Therefore, the degrees in the weaved graph have to be corrected by re-calculating them. After this has been performed, the resulting Multimodal Spatial Network is stored in the database. It could already be exported and used for isochrone computation. Since algorithms and enhancements require additional data or include preprocessing steps, another step is carried out.

7.4. Dataset Optimization

The dataset optimization is an optional step, that performs multiple things:

- Dataset enrichment with elevation data
- Preprocessing of average de- and inclines to speed-up elevation awareness
- SpiderWebGraph generation to allow for traversing of areas/places
- Materialized view creation to speed up computations
- Vertex density computation used by MineR(X) algorithms
- Transforming the schedule representation to an interval-based model

- Tile table creation to allow MineT(X) algorithm usage on non-spatial-databases

The actions address multiple issues, which will be discussed in the upcoming two sections.

7.4.1. Elevation Data

Information about elevation needs to be added to the network in order being able to compute user profiles and to allow for elevation aware isochrone computation. Data about elevation is available in form of Digital Elevation Models (DEMs). Research about them has been carried out together with Silbernagl et al. [114]. Although there are many DEMs available, only some of them are freely available. Less are of global, or at least near global, coverage. However, a DEM has been created within a eleven day mission of the space shuttle Endeavour in February 2000. It covers the entire surface of the earth only excluding the polar regions. The resulting data is known as the Shuttle Radar Topography Mission (SRTM) dataset. Since the result misses information in some locations, e.g. due to reflections or objects blocking the radar, the data has been postprocessed with several different approaches. SRTM is available in two different resolutions. The dataset with lower resolution records points with assigned elevation every three arc-seconds (equaling to approximately 90 meters). The dataset has been released into public domain, allowing others to develop approaches to correct voids and to improve data. Such an improvement is developed by the Consultative Group for International Agriculture Research known as CGIAR dataset. However, there are multiple versions of the datasets, for example CGIAR v4 or SRTM v4.1. Each version includes improvements, like equaling the level of the oceans, resulting in the fact that not every improvement on the original dataset produces a better dataset than the next version of the original SRTM data. In addition to the three arc-second dataset (known as SRTM-3), at the end of 2015 another one arc-second dataset (known as SRTM-1) was released into public domain. Until then this resolution (which equals approximately 30 meters) was only available for the United States.

Data from SRTM-3 and SRTM-1 datasets are applied in the latest version available to the Multimodal Spatial Network that was created in the previous steps. Doing this every vertex and hypervertex are enriched with elevation information. Interpolation is applied to be as exact as possible and to not solely rely on single values from the dataset. There remains a major problem that needs attention. If a street in the network is very long and without curves, only the start and end vertex could be modeled. There is simply no

need to use hypervertices that are used for geometric representation. Without the hypervertices a hill in between could simply be missed. This has also been explained in Section 6.3 when introducing the Nyquist frequency, but also needs to be regarded during dataset creation. Therefore, edges are extended with intermediate hypervertices, allowing to add elevation information without missing hills on long, straight streets.

A similar approach that OpenStreetMap is to geographical data, is OpenDEM [100]. It improves data from the SRTM dataset by using contextual information from OSM that is applied to the digital elevation model. For some regions, where there are official DEMs freely available from governmental or other institutions, OpenDEM integrates such information. However, although it builds on data from SRTM, it is not of global coverage. This and the fact that various data sources are included make it impossible to compare computation results across the globe.

After elevation data has been integrated into the Mulimodal Spatial Network, an elevation aware computation can be performed with the methods described in Section 6.3.

7.4.2. Computed Information

Multimodal Spatial networks can be extended with information that is created by preprocessing steps. From the elevation data that has just been added to the database average values for declines and incline can be computed. By further adding the distance on which an edge inclines (in percent), elevation aware computation can be simplified. Instead of computing the slope between every hypervertices within the network, the computation has only be done once per edge (which typically includes multiple hypervertices).

A problem arises due to the manner OSM models places, such as the central square in a city. There must not exist edges that connect the streets starting (or ending) in such a plaza. Instead, the area is tagged by a separate metadata field, marking it as polygon that should be used for routing. While some algorithms only use the borders of these polygons, a more sophisticated solution can be realized when utilizing an additional preprocessing step. During this step a so called “SpiderWebGraph” is generated [27]. The resulting continuous edges and vertices are merged with the Multimodal Spatial Network, allowing all algorithms capable to compute isochrones to route over polygons.

To further speed up computation materialized views can be used in the database. For the algorithms MineR(X) and MineT(X) database tables need

to be joined. When using a materialized view that represents the joined tables, the time needed for the join can be shifted into dataset creation. This increases the file size of the dataset, but allows faster computation.

The vertex density table, that is necessary when limiting ranges that are loaded by the MineR(X) algorithms and which has been described in Section 4.2.3 and Table 4.1, is also created in the dataset optimization step.

If a database that does not include spatial database functions is utilized to store the datasets, the algorithms MineR(X) and MineT(X) can not be used. This can not be changed for the range loading MineR(X) algorithms, but if during dataset creation a spatial database is used, an additional database table can solve this issue for the MineT(X) algorithms. A simple table assigning every vertex to a tile at various zoom levels eliminates the need for the spatial functions. With the resulting table that contains the columns $(vertex, zoomLevel, tile)$ it then is possible to load all vertices within a tile without the need to create bounding boxes and without using containment or intersection operators.

Additional transformations of the network are also performed in this step of the dataset creation workflow. As mentioned in Chapter 4 the underlying representation of the schedules should be changed from an instance-based representation to an interval-based model. This modification is applied during this optimization step before it is exported.

7.5. Exporting and Providing Datasets

After the four previous steps have successfully been performed, a valid and preprocessed Multimodal Spatial Network is stored inside the database. It can be either postprocessed or exported directly by using database functionality. The postprocessing step has been used throughout several bachelor and master theses which were carried out at the University of Innsbruck that evaluated isochrone computation runtimes across various databases and database types.

Besides a PostGIS enabled PostgreSQL database also SpatiaLite and Neo4J-Spatial seem suitable to compute isochrones. SpatiaLite can be easily used to keep datasets entirely in memory allowing fast computation after initial loading. Although this delivers great performance for the queries used by Mine(X), performance drops for the MineR(X) algorithm and when regarding schedules of the public transportation system SpatiaLite can not keep up with PostGIS.

In addition, the performance of SpatiaLite suffers as soon as spatial functionality is used [107]. The performance of isochrone computation within the graph database Neo4J-Spatial is not as good as it is in PostgreSQL/PostGIS. Depending on the access mode, it is usable (in embedded mode when using the Cypher query language), but still not as fast as PostGIS [11]. However, by further optimizing the graph representation for Neo4J-Spatial or by using direct access to the stored graph, e.g. by using the query language Gremlin instead of Cypher, performance could still be improved.

To allow comparison across different research groups and to ease using Multimodal Spatial Network for various purposes, some datasets that were created with the process described in this chapter can be downloaded from the University of Innsbruck [59].

CHAPTER 8

The Application IsoMap

Isochrones and the reachable areas describe locations that are reachable with or within a certain time duration. Therefore, the results can be visualized. The graphical representation adds information that otherwise would be hard to record and eases interpretation of the outcomes. As the proverb says, a picture is worth a thousand words. Therefore, a graphical user interface has been realized that serves as prototypical implementation to view isochrone computation results. It has been named "IsoMap", since it shows a reachable area together with the isochrone on top of an interactive map. The web application is freely available to everybody on `https://dbis-isochrone.uibk.ac.at` [60] or as source [61].

IsoMap is not the first attempt to draw isochrones on top of a map. The web-based system "ISOGA" allows for reachability analyses as well [52]. Another user interface named MineX (like the algorithm) used to introduce the MineX algorithm in 2012, was developed. ISOGA (and MineX) did not match expectations regarding performance, robustness and usability. Therefore, a new approach has been implemented.

8.1. System Architecture

Like ISOGA, the new application IsoMap builds on web technologies and a three-tier architecture. By design, IsoMap is much more flexible when compared to ISOGA. Although IsoMap still is capable of using OGC standards for communication, newer techniques have also been realized. The Javascript Object Notation (JSON) can be used for communication through a REST-API or with websockets. Spatial Reference Identifiers (SRIDs) are handled correctly by the system for the geographical information. Whenever communicating with the backend, coordinates are given in SRID "EPSG:4326", while every valid SRID known to the database can be used within the database. To increase performance, web optimization technologies like minification on Javascript (JS), HyperTextMarkupLanguage (HTML) and Cascading Style Sheets (CSS) were applied. In addition, compression is applied using the Deflate algorithm, keeping the processing of large isochrone results fast.

Results can either be sent to the client using a Web Map Tile Service (WMTS) provider (like Geoserver) or by using GeoJSON. GeoJSON itself uses the Well-Known-Text (WKT) representation to describe geographic objects. The difference between using WMTS and GeoJSON is, that WMTS information is created on the server. Results are then sent as rastered images to the client. In contrast, GeoJSON sends the vendor based result information directly as text, allowing result manipulation by the client. When it comes to online maps, GeoJSON can be of benefit, since zooming the map can be done without contacting the WMTS server. When using the OGC standard, information would have to be transferred for every zoom level.

The resulting three tier architecture consists of the following elements:

- A spatial database (in the Data Tier)
- The servlet container that uses REST and HTML websockets. It also delivers JS, HTML and CSS files to clients (in the Application Tier).
- An OGC compliant WMTS server (in the Application Tier).
- Web browsers running on various platforms, such as desktop systems, laptops, smartphones and other mobile devices (in the Presentation Tier).

8.2. Operating Principle

When opening the IsoMap web application, three files are sent to the client. The website itself, as HTML-file. Then the design information (as minified CSS-file) and at last the behavior information (as minified JS-file). Afterwards, the client executes the main function in the JavaScript file which opens a HTML5 websocket connection to the webserver. Next, configuration information is requested by the client. On this request, the server assigns an identification number to the client and sends information about the datasets that are stored in the database. Also the user profiles that are available are determined and sent to the client. Since an interactive online map is used, some more requests will be sent to tile servers delivering the so called "base map" (which is provided by an OpenStreetMap service for example). IsoMap itself includes advanced caching of the map tiles in a browser-enabled database (a PouchDB instance), so that communication with web map tile services is reduced to a minimum.

After this initial loading step has been carried out, the user in front of the web browser now is able to search for locations on the map either by using geocoding functionality (name to latitude/longitude) provided by IsoMap or by browsing the interactive map. When selecting a location it is analyzed for isochrone computation suitability. In particular, this means that the datasets which have been retrieved with the configuration object before, are checked for inclusion of the selected point. This is done with two operations and has been inspired by spatial database inclusion checks. First, the bounding box of all the datasets is used the check for location inclusion. This can be done without any further communication to any server, since the bounding box information is included in the configuration. If matching datasets are found, additional information is requested from the server. The information equals the boundary polygon of the dataset. This boundary polygon is the same that has been used on dataset creation which was described in Chapter 7. Only if the selected location lies within the bounding polygon, the location is suitable for isochrone computation. One of the datasets the point is located within can then be selected and used for isochrone computation.

If the selected point is suitable, then the user has to enter a starting (or ending) date and time as well as a maximal travel duration. This information is used together with some configuration properties and send to the server. These properties can be changed in a separate dialog and include:

- *Algorithm*: The algorithm that should be used for computation. Includes all the algorithms described in Chapter 4 and Chapter 5. The list

covers the eight algorithms MDijkstra, Mine, MineX, MineR, MineRB, MineRX, MineT and MineTX.
- *Network mode*: Sets the method how the underlying datasets graph is treated. Besides "Isochrone (unimodal)" and "Isochrone (multimodal)" also "Isodistance" or "Automatic" can be chosen. The automatic mode changes between unimodal and multimodal isochrone computation, depending on the dataset.
- *Direction*: Describes in which direction the computation is performed. Can either be "Incoming" (to the query point) and "Outgoing" (from the query point).
- *Speed estimation*: Lists the approach that is used to estimate speeds. It includes all the methods explained in Section 6.3 and also the available user profiles. At least the methods "Bike", "Bike (+elevation)", "Walking", "Walking (+elevation)", "Fixed", "Simple", "Simple (+elevation)" and "Automatic" can be selected. The difference between the fixed and the simple modes is that the fixed mode always uses the same speed on every road, while simple applies an upper bound. The automatic mode either uses the walking mode or the elevation aware walking mode depending on the fact, if the dataset the isochrone is computed in, includes elevation information.
- *Enclosure*: The algorithm used to create the isochrones enclosure. Besides server based methods also some methods computing the result on the client (in JavaScript) are available.
- *Expiration mode*: A flag that enables visualization of the vertex states instead of showing information about the public transportation system.
- *MultiIsochrone*: A flag that allows to compute multiple isochrones without clearing a previous result. This can be useful when searching for locations that are reachable from multiple query points.

After the server received the information, it starts to compute the isochrone. Depending on the dataset, the maximal travel duration, the configuration and the system setup, the duration can last from some milliseconds to multiple seconds and even minutes. This is where IsoMap provides a significant advantage over the ISOGA and MineX. Since websockets are used, the application does neither use polling techniques (such as long polling) that increase the server's load, nor does it need to worry about possible timeouts.

The result can either be sent as GeoJSON object or with the help of an WMTS server. Using a Web Map Tile Service gives the advantage that information is customized to the current zoom level of the map. This means that for very large results, e.g. the reachable area covers all the middle of Europe, only some (relatively small) images are transferred. If the same result would be sent as vector-data, more bytes would have to be transmitted. The opposite

effect occurs for small results. In such cases transferring images always results in transferring more bytes than just some small GeoJSON data. This could be changed if the vector data would adapt to the zoom level on the client. However, this behavior is not implemented in IsoMap, since it would also increase the number of requests when navigating the interactive map.

8.3. Features of IsoMap

IsoMap provides several functionalities. The most important ones will be shortly explained in the next sections.

8.3.1. Visualization of Isochrones

The data that has been sent to the client is visualized with layers. Depending on the configuration used, different layers are available. For MineR(X) or MineT(X) information about the loaded ranges/tiles is available. For all algorithms the reachable area, the reachable ways and information about transit stations is available. Besides the computation results, also a base layer (the lowest level of the map) and a hill shading layer can be activated using client functionality. IsoMap visualizing the default OpenStreetMap base layer and a computed isochrone is shown together with all available layers in Figure 8.1

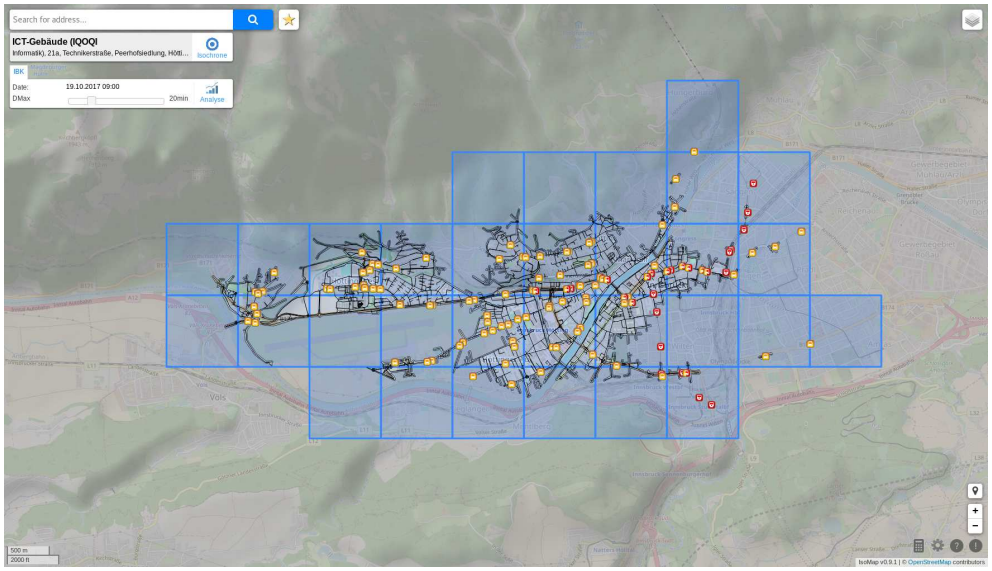


Figure 8.1.: IsoMap showing all available overlays.

By default, IsoMap shows the result after the calculation has finished. Moreover, there is a prototypical implementation available, that plots the result during computation. This way of visualization is called "incremental visualization" and has been researched together with J. Winder throughout his Bachelor thesis [129]. It became obvious that although incremental visualization can be usable for very large results, the major problem is the computation of the borderline after the graph of all reachable locations has been created. In terms of the definitions given in Chapter 3, the problem is to compute the isochrone, not only the reachable area. Therefore, multiple algorithms have been implemented that create the isochrone from the reachable area. They can be chosen with the parameter "Enclosure" in the configuration dialog.

The base layer is the only one that can not be hidden. The visibility of all the other layers can be toggled. That is why all those layers are called overlays. However, even the base layer can be exchanged. Available base layers in IsoMap include various derivatives from OpenStreetMap, e.g. OSM Hot and OSM DE. In addition, base layers from Microsoft Bing and even Google Maps can be used. For the region of Austria the BaseMap, including governmental data, can be chosen from the list of base layers. In Northern Tyrol also layers from the Tiroler Rauminformationssystem (Tiris) are available.

8.3.2. Vertex Expiration Information

There is a special visualization mode available that allows to visualize the states of the vertices that have been used during computation. It is called "expiration mode" since its initial use was to understand vertex expiration that has been explained in Chapter 4. If this mode is used additional information can be figured for all the vertices. Besides the information to which disjoint expiration group the vertex belongs (either O, C or X) also graph's vertex id and the time that was needed to reach the vertex as well as the remaining duration is shown.

In the Figure 8.2, the white vertices correspond to the expired group X, the gray ones belong to the group C. The black ones have been loaded, but not expired and therefore are part of the open group O.

8.3.3. Public Transportation System Usage

To gather further insights and to better understand how the result was computed, information about the public transportation system can be explored

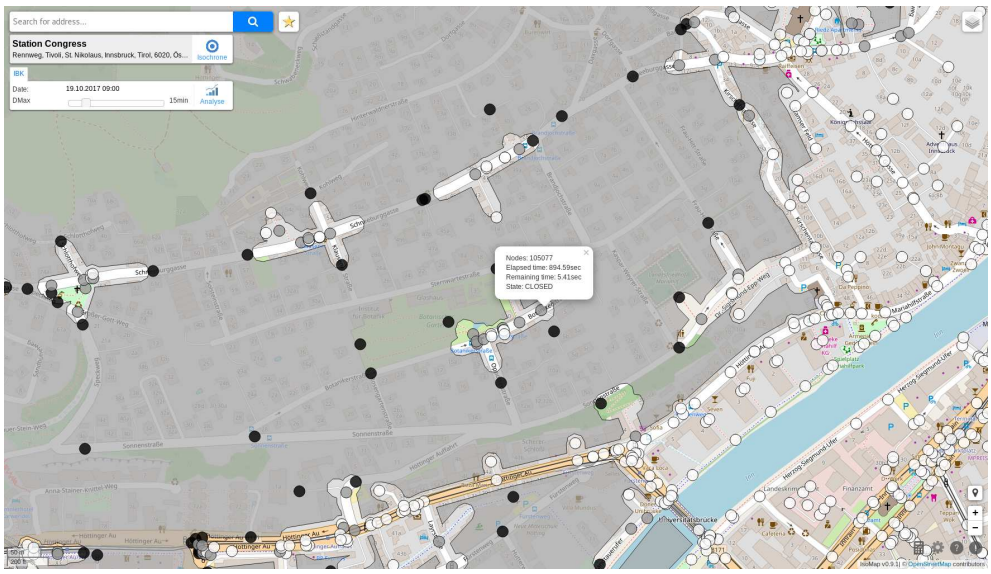


Figure 8.2.: Vertex expiration mode in IsoMap.

within IsoMap. For each station, a contextual menu can be accessed that allows to view departure (or arrival) times of bus routes. Although routes are stored in order to display their names, the course of a route is not displayed as part of the isochrone. However, IsoMap is able to illustrate the course using information provided by the OSM transport base layer. Data about the course of a route is stored in OSM, although the schedules are not recorded. The contextual menu together with the transport base layer is shown in Figure 8.3.

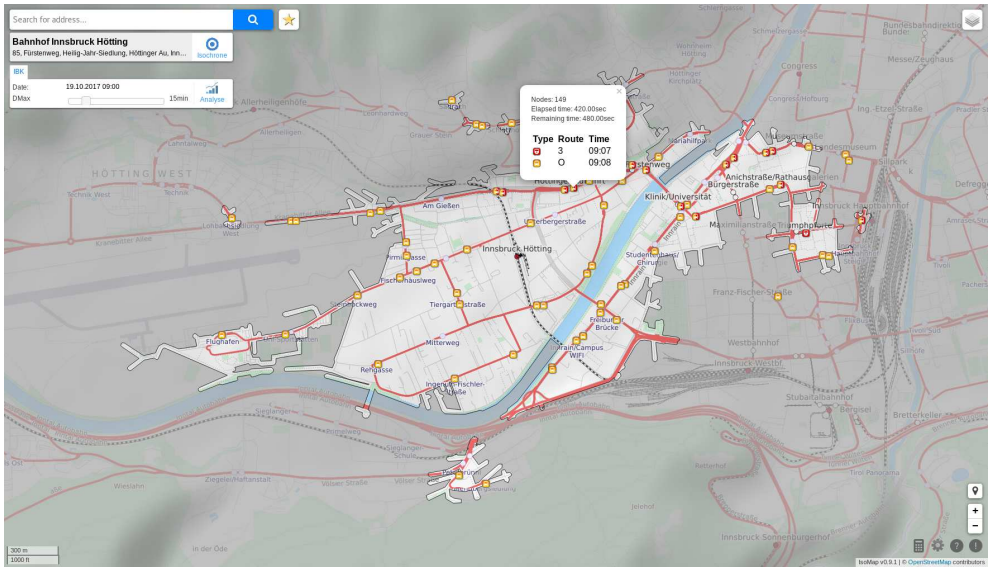


Figure 8.3.: IsoMap showing information about transport systems.

8.3.4. Spatial analyses

IsoMap is able to request additional information for visualized results from external sources. Information from OpenStreetMap can be combined with the result and overlaid with the calculated reachable area and the resulting isochrone. This way spatial analyses can be carried out within IsoMap.

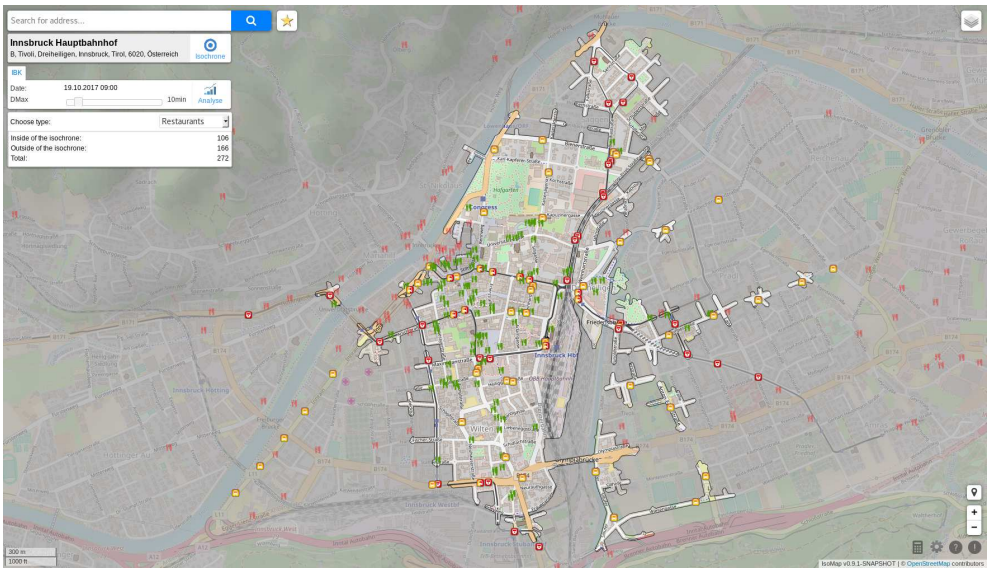


Figure 8.4.: Spatial analysis carried out in IsoMap.

In Figure 8.4 an isochrone is visualized that is further used to determine the number of restaurants that are reachable within ten minutes when starting from the query point. The result is shown in the top left corner, stating that out of 272 restaurants in the city of Innsbruck, Austria a total 106 restaurants are reachable from the main station within ten minutes (when starting on the 19th of October in 2017 and using public transport).

Similar analyses are also possible for schools, universities, nursery schools and colleges. Hierarchical grouping of analyzable facilities is also possible, so isochrones can be checked for inclusion of "educational buildings". More categories could easily be added to IsoMap, but for demonstration purposes only educational facilities and restaurants have been implemented.

8.3.5. Averaged and Time-invariant Isochrones

In Chapter 5 averaged isochrones and time-invariant isochrones were introduced. To allow visualization of the results created by these enhancements, a

separate tool has been integrated into IsoMap. Because the geometry of intermediate results are used for processing, it is termed "geometryCalculator" and is available in the map's plugin list at the bottom right. After selecting a query point, the tools utilizes a dialog to asks for some settings. In addition to a time span (given by starting date and time as well as ending date and time), the mode and the number of intermediate results have to be selected. The mode allows to select "Averaged isochrone", "Time-invariant isochrone" and the combination "Averaged and time-invariant isochrone".

After all settings have been entered, the tool is able to compute the intermediate results and sends information about them to the client. The information is then used to create the result referred to by the mode. The result actually is the thick black border, whereas the single intermediate reachable areas are overlaid in green with a transparency of 25%. The example used for explaining averaged isochrones throughout Chapter 6 is shown in Figure 8.5. Time-invariant isochrones would be shown in a bluish color.

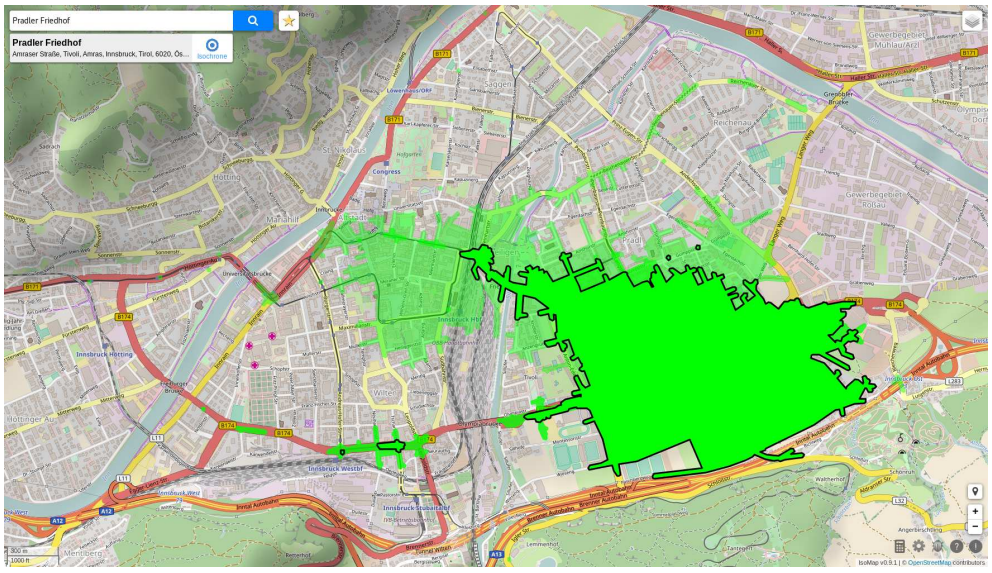


Figure 8.5.: Averaged isochrone in IsoMap.

8.3.6. Traversing of Areas/Places

As already mentioned in Section 7.4.2, OpenStreetMap allows to model plazas with polygons. To allow movement across polygons some preprocessing is needed. During this step, a so called "SpiderWebGraph" is created as defined by Dzarfic [27]. Since IsoMap is able to show a separate layer for the traversed edges, these graphs can be easily visualized. Figure 8.6 shows a SpiderWebGraph laid over the Piazza Walther in Bolzano.

Figure 8.6.: SpiderWebGraph in IsoMap on Piazza Walther, Bolzano, Italy.

CHAPTER 9

Evaluation

In this chapter a detailed empirical evaluation of the algorithms that have been discussed throughout this thesis is given. This is done with the help of two synthetically generated and four real-world datasets and in a similar fashion to the empirical evaluation that was performed by M. Innerebner [51].

After describing the test system and the datasets, the data structure that has been used for the vertex expiration queue is examined at first. Then, the best adaptive zoom levels for the MineT and MineTX algorithms are determined. Evaluations regarding the runtime and the memory consumption are given together with so called "break-even-points" that describe how long algorithms based on incremental expansion stay faster than the in-memory algorithm MDijkstra.

9.1. Evaluation System

The system used for the evaluation consists of an Intel® Core™ i7-6700K combined with 32 GB of DDR4-3200 memory. The base frequency of the 64bit quad-core CPU is 4.0 GHz, while it allows for eight threads in parallel (using

hyper-threading) and a maximum frequency (in turbo mode) of 4.2 GHz. The Z-170 Intel chipset from the mainboard connects to a 256 GB Samsung 950 Pro NVMe M.2 SSD disk drive on which Fedora 26 running on a Linux Kernel 4.13 is installed. Together with PostgreSQL 10 and the PostGIS extension (used in version 2.4.0) a complete spatial database setup is provided. Like it is true for the MineR(X) algorithms, the MineT(X) algorithms also make use of geospatial functions within the database and do not include a preprocessed tile information table that was discussed in Section 7.4.

To minimize the effects of outliers caused by the operating system itself, the tests listed in the following subsections were carried out 25 times. All the plots in this chapter represent the median of the results of those test runs.

9.2. Datasets under Test

The datasets that were used for evaluation purposes have all been generated in October 2017 with the tool "osmPti2mmds" that was described throughout Chapter 7.

The synthetically generated datasets refer to a grid based layout and a spider layout. The underlying networks are similar to the ones figured in Figure 4.1(b) and in Figure 4.1(c). The grid network has an edge length of 100 vertices, while the spider networks has six different axes with 1000 vertices on every edge. The Spatial Reference Identifier (SRID) for these datasets is "EPSG:3857", which models the plain mathematical models. The statistics for the synthetic networks used during evaluation are given in Table 9.1.

Dataset	Size	$\|V\|$	$\|E\|$	$\|E_{csct'}\|$	$\|E_{dsdt'}\|$	$\|S\|$
Grid network	12.52 MiB	10 k	39.6 k	39.6 k	0 k	0 k
Spider network	8.39 MiB	6 k	24 k	24 k	0 k	0 k

Table 9.1.: Statistics about the synthetic datasets

The synthetic datasets consist solely of continuous space and continuous time edges. Therefore, the computations are all performed in unimodal fashion for those two. In contrast to the synthetic datasets, the real-world datasets are true Multimodal Spatial Networks containing continuous and discrete edges. The SRID of the data is "EPSG:4326" (also called WGS84), that is used by the Global Positioning System (GPS) to represent arbitrary places on the Earth

using latitude and longitude. Statistical measures of the datasets are given in Table 9.2.

Dataset	Size	$\|V\|$	$\|E\|$	$\|E_{csct'}\|$	$\|E_{dsdt'}\|$	$\|S\|$
BER	1569.84 MiB	223.31 k	716.4 k	576.24 k	15.92 k	3074.66 k
BZ	42.89 MiB	6.58 k	17.78 k	16.48 k	0.72 k	49.35 k
SF	256.43 MiB	34.65 k	99.93 k	87.07 k	5.77 k	767.24 k
VIE	775.31 MiB	120.16 k	335.52 k	319.75 k	6.88 k	1354.95 k
WDC	301.45 MiB	41.01 k	122.78 k	109.02 k	7.33 k	698.52 k

Table 9.2.: Statistics about the real-world datasets

The datasets represent the cities of Berlin (BER), Bolzano (BZ), San Francisco (SF), Vienna (VIE) and Washington, D.C. (WDC). The reasons for this selection are simple. Berlin and Vienna have been included, because these two transportation companies in these cities offer schedules in GTFS format. They model larger cities in Europe, so that computation runtimes are expected to be large. The datasets of San Francisco and Washington, D.C. represent the counterpart in the United States. It is believed that cities in Europe model a more spider-shape network and cities in the U.S. use a grid-based model. Therefore, cities from these two continents are included. The small city of Bolzano is used to allow comparison with former publications by Gamper et al. and also to include a smaller dataset. While the four bigger datasets have been created directly from GTFS files that were supplied by the public transportation companies, information about the buses used in Bolzano was converted from VDV452. However, this process is officially suggested by the company "Società Autobus Servizi d'Area (SASA)" [117] that provides the data as open source.

9.3. Choosing the Data Structure

The first evaluation compares different data structures used for the queue that holds the open vertices (group O). The reason why this is important has been discussed in Section 5.1 and is now evaluated. The following computations were performed with the MineR algorithm, since it produces a heavy load on this data structure. In contrast to Mine(X) a lot of vertices are kept in O, while regularly a bunch of additional vertices is added (in contrast to MDijkstra, which holds even more vertices in O). In the following the results for the datasets BER, VIE, SF and WDC are given. The synthetic datasets as well as BZ are skipped, since the amount of vertices kept in O is too small to see a difference in runtime, no matter which data structure is used. In

Figure 9.1 the results for the four datasets listed above are given. On the x-axis the limiting criterion for the multimodal isochrone computation, the maximal duration d_{max} in minutes, is plotted, while on the y-axis the computation time (in seconds) is used.

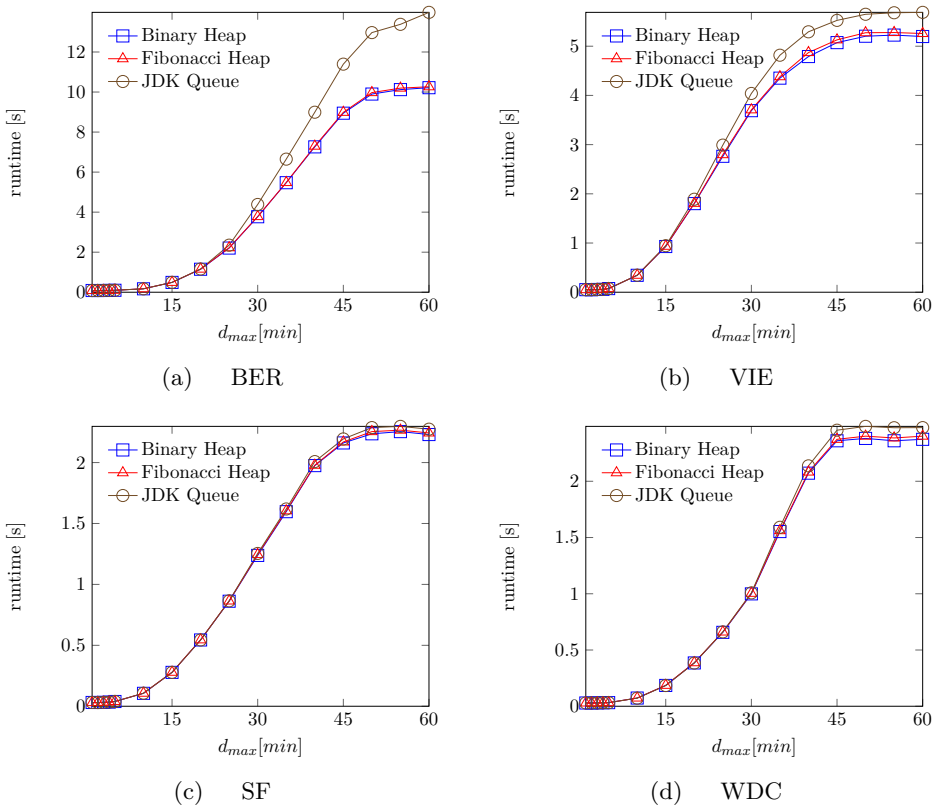


Figure 9.1.: Comparing expansion queue data structures.

Especially in the European datasets of Berlin (BER) and Vienna (VIE) it can be observed that the queue of the Java Development Kit (JDK) is the slowest. Although it uses a JDK internal class, i.e. java.util.PriorityQueue, and a balanced binary heap, it performs worst in all of the datasets. The reason for this might be that there is no possibility in the API of the class to directly change the order of an element in the queue. On decreasing the network distance of a vertex it has to be removed and re-added with the updated distance. The two other implementations allow direct manipulation of the order in the list and perform better. For the dataset of Berlin and a d_{max} for one hour the computation time of the Fibonacci Heap implementation is only 73,4% when compared to the JDK's PriorityQueue. The runtime of the Binary Heap implementation is even faster and only 73,0%. However, the difference between the Fibonacci and the Binary Heap is only 50 milliseconds for a computation time of a little over 10 seconds. For the dataset of Vienna

results look similar, but with a slight improvement for the Binary Heap when compared to the Fibonacci Heap. While for Berlin the runtime of the Binary Heap is 99,6% of the Fibonacci Heap in Vienna it is 98,8% (a difference of 59msec on a total computation time of around 5.2 seconds). In the datasets of the United States, all the implementations are quite fast, although the same order of the implementation holds. The fastest is always the Binary Heap implementation, followed by the Fibonacci Heap and the JDK internal class. For San Francisco all the implementations stay within two percentage of each other, while in Washington, D.C. the Binary Heap takes up 95,8%, while the Fibonacci Heap's runtime is 96,9% of the JDK queue.

It becomes obvious that the performance of the JDK queue gets worse for larger computation times that are caused by larger datasets. Although both, the Fibonacci Heap and the Binary Heap implementation, perform better, there is a slight advantage for the Binary Heap implementation. When looking at the complexities of the MDijkstra algorithm, this can be explained by the reduced complexity of the algorithm when using a Binary Heap when compared to the complexity when using a Fibonacci Heap.

As a result of this evaluation step, all the further steps will be carried out with a Binary Heap.

9.4. Determination of the Best Loading Ranges

During discussion of the MineR(X) algorithms in Section 4.2.3 the limitation of the loaded ranges by memory was mentioned. Therefore, it is analyzed whether choosing ranges according to the remaining duration or always loading a fixed number of vertices performs better. Calculations are performed for vertex densities of 1000, 2000, 3000, 4000, 5000 and without limitation applied. The results are given in Figure 9.2. On the x-axis the maximal duration d_{max} in minutes is given, while on the y-axis the computation time (in seconds) is used.

Although it is quite difficult to examine differences between the different range limitations in the figures, one observation can clearly be made. MineR without limiting the ranges performs better than when using a fixed number of vertices. The reason for that could be that by joining the precomputed vertex density information takes too much time or that limiting the ranges itself is not of great benefit. Therefore, range limitations should be avoided whenever possible. They are not applied during the evaluations carried out throughout the rest of this chapter.

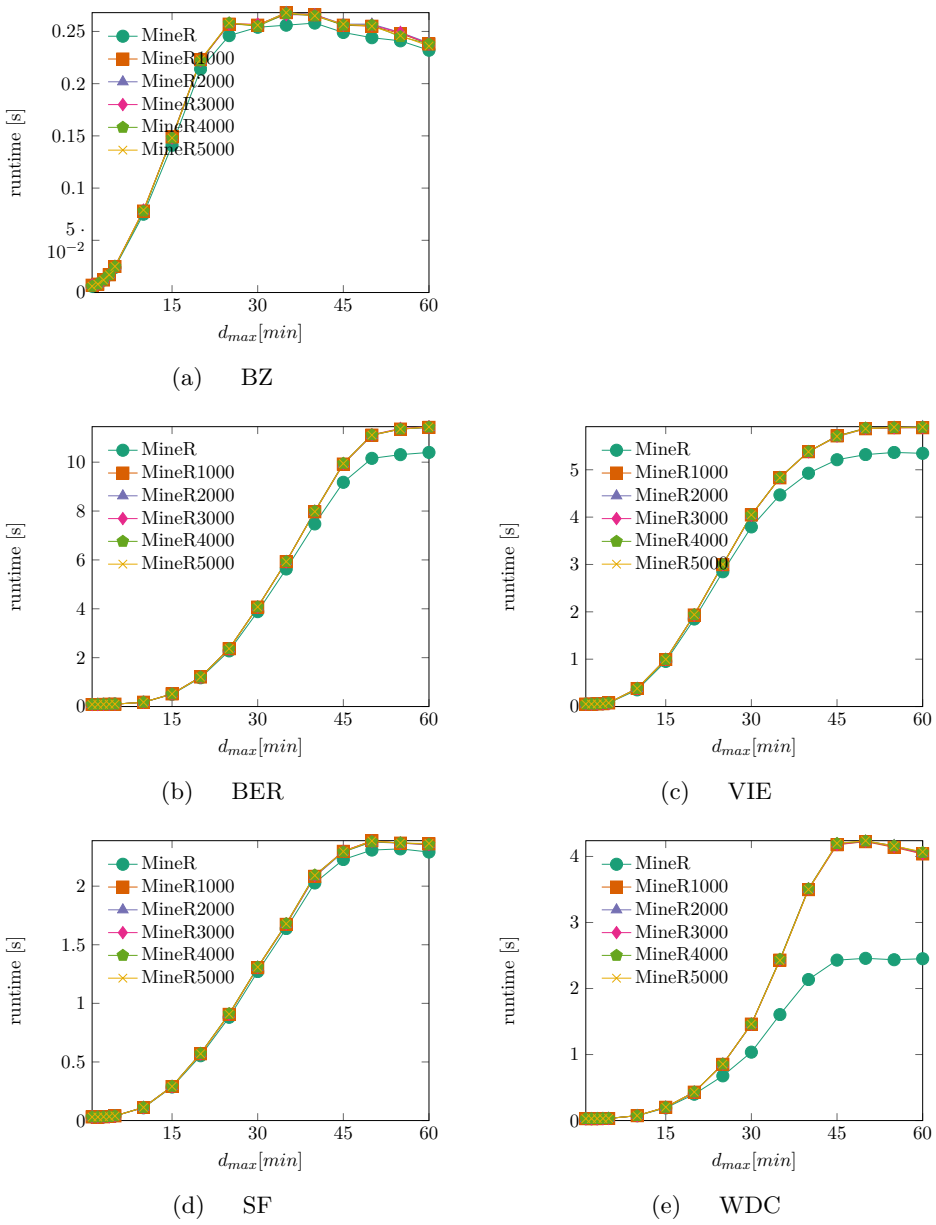


Figure 9.2.: MineR range limitation performance.

9.5. Determination of the Adaptive Tile Ranges

When it comes to the MineT(X) algorithms adaptive zoom levels are of great benefit. Their determination is done by computing the computation time of various fixed zoom levels z and then using the zoom level with the best result.

This has been described in Chapter 5 and is now evaluated. The computation times on the real-world datasets are plotted in Figure 9.3 up to a maximal duration of one hour. Tile sizes from 12 to 18 are evaluated. This equals a tile region edge length of 9775,87 meters on zoom level 12 and 152,576 meters for z equal to 18 (on the earths equator). The x-axis again represents the maximal duration d_{max} in minutes, while on the y-axis the computation time (in seconds) is shown.

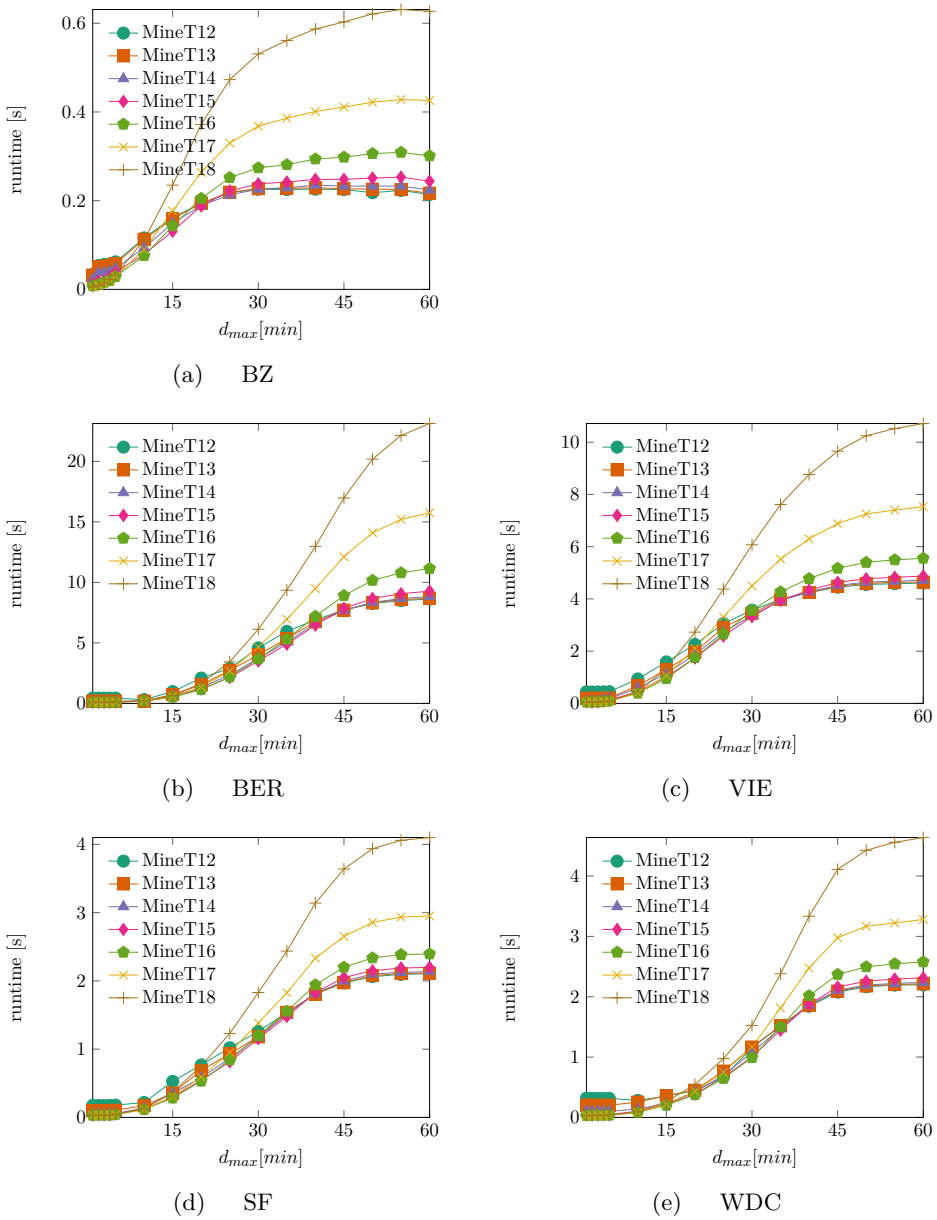


Figure 9.3.: Determining adaptive tile ranges in real-world datasets.

For all the datasets it can be seen that the optimal zoom level depends on the maximal duration d_{max} of the isochrone. For small durations also smaller tile regions are of benefit while for larger durations the tile region size should increase (meaning that z gets smaller). For very large tile regions the initial loading range would load the entire graph of the dataset, making its memory consumption equal to MDijkstra. However, the best zoom levels can be determined by simply selecting the best runtimes for every d_{max}. Since for larger durations it is hard to see the exact numbers from the figure, the results are also given in Table 9.3. Every line corresponds to the dataset under test. Every column equals a zoom level z and the value in the cell represents the duration where the zoom level should be used.

Dataset	z=18	z=17	z=16	z=15	z=14	z=13	z=12
BER		0	333	1355	2531		2759
BZ		0	264	660	1230		1833
SF	0		210	1311		2324	2460
VIE		0	120	1157	2037		2280
WDC		0	665	1859			2304

Table 9.3.: Best adaptive tiles ranges in real-world datasets.

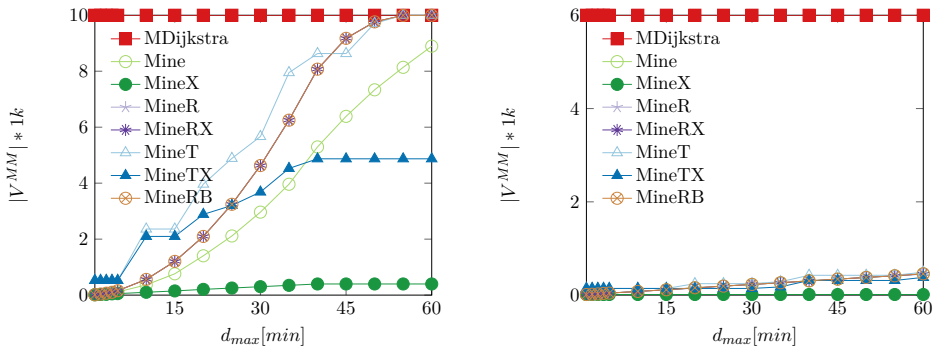
For the dataset of Berlin for example, Table 9.3 reads as follows. Between remaining durations of 0 and 332 seconds it is best to use zoom level 17. From 333 to 1354 seconds larger tiles of zoom levels 16 perform best. Then up to a duration of 2530 seconds a z of 15 is preferable. For durations between 2531 and 2758 seconds tile ranges with a zoom level of 14 provide the fastest computation. For all the larger remaining durations a zoom level of 12 should be used. For the latter ones the whole dataset of Berlin is contained in approximately 50 tile ranges.

For the following evaluations and the MineT(X) algorithms the adaptive zoom levels given in Figure 9.3 and Table 9.3 are used.

9.6. Memory Experiments

After all the best loading ranges as well as adaptive tile ranges were fixed, evaluations regarding the algorithms memory and runtime are carried out. The figures present the memory that has been used at maximum throughout the computation. The results for the synthetic networks are given in Figure 9.4. The x-axis shows the limiting criterion d_{max} in minutes, while on the y-axis

the number of vertices kept in memory (maximal value throughout the entire isochrone computation) is plotted.



(a) Grid (b) Spider

Figure 9.4.: Memory consumption in synthetic networks.

MineRB loads the same ranges as MineR(X) in batches. Therefore, no difference can be seen for memory consumption. For both algorithms a single range query is sufficient to load everything into memory that is needed for isochrone computation. This initial loading range caused the algorithm to use a lot of memory in synthetic networks. MineTX loads a bit less vertices into memory, especially in the grid network. The algorithms using vertex expiration free memory early and therefore consume less memory.

For the real-world datasets that contain discrete time edges and are true Multimodal Spatial Networks the memory consumption is given in Figure 9.5.

It becomes obvious that MDijkstra performs the worst, since it loads the whole network into memory. The algorithms not using vertex expiration start with a low memory consumption that reaches an upper bound. This bound is caused by the fact that the entire network needs to be considered on isochrone computation. That also means that within an hour big parts of the dataset are reachable. There are minimal advantages of Mine over MineR and MineRB, whereas MineT consumes the most memory except for MDijkstra. MineRB behaves exactly like MineR also in the Multimodal Spatial Networks. MineRB reduces the number of database calls, but the loading ranges are exactly the same, even when using discrete edges. More insight is given for the algorithms with vertex expiration. As can be seen, Mine consumes by far the least memory. This is what was expected, since Gamper et al. showed that in terms of memory MineX is optimal [34]. MineRX is somewhere in between MineX and the algorithms not using vertex expiration. This has been shown by M. Innerebner and was also expected [51]. The algorithm MineTX, that has been discussed in Section 5.3 performs great. It uses adaptive zoom levels

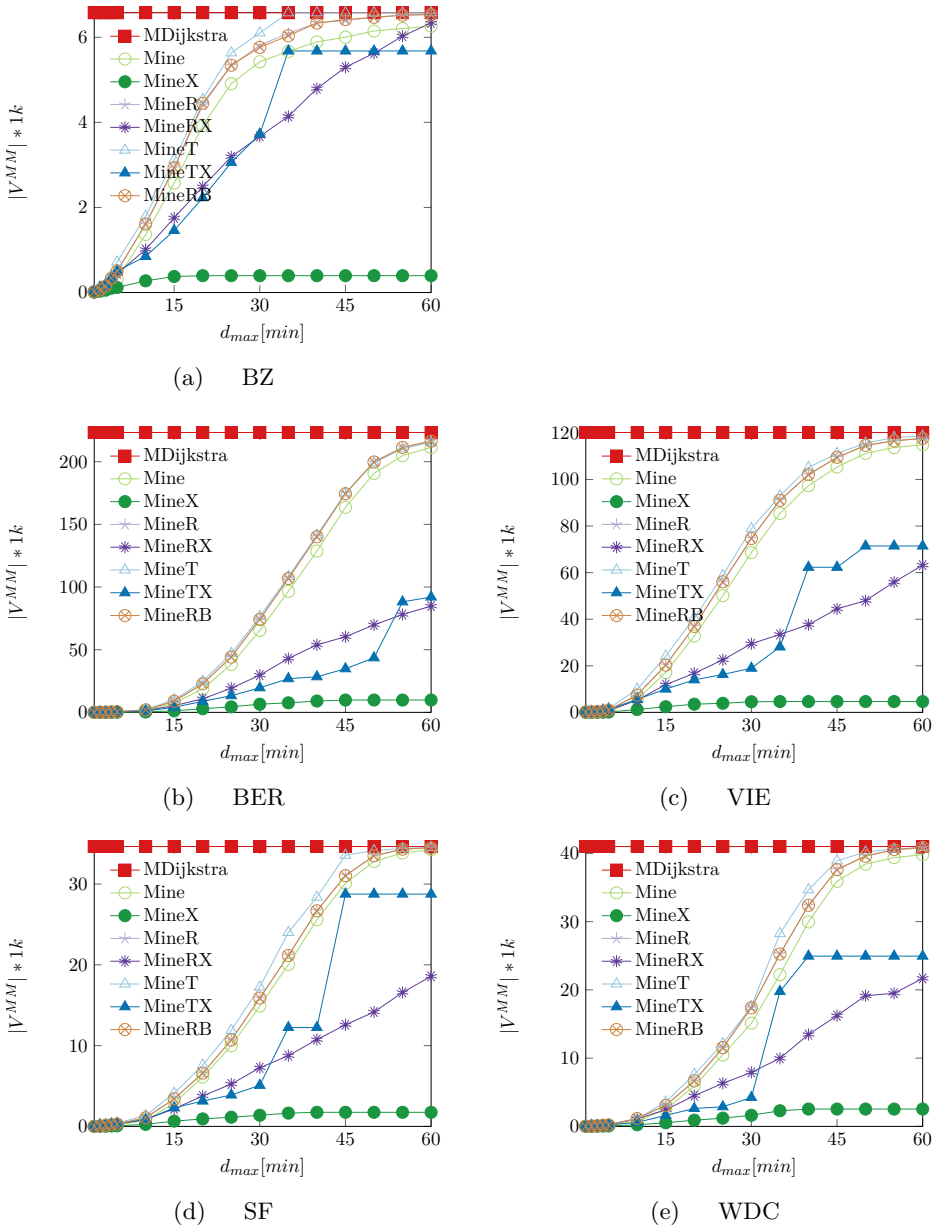
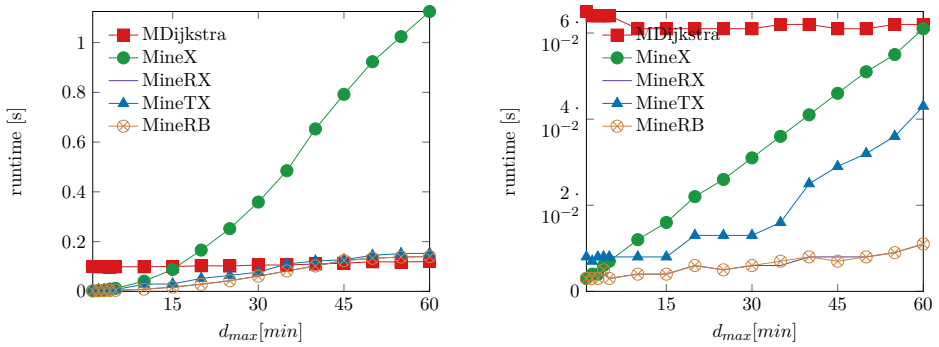


Figure 9.5.: Memory consumption on real-world datasets.

and therefore introduces spikes. On every change of the zoom level memory consumption increases very fast, but this allows to keep the performance high. This can be best seen in the dataset of San Francisco, where two such spikes occur for a d_{max} of 35 and 45 minutes. Depending on the dataset and d_{max}, it can be seen that sometimes MineTX and sometimes MineRX performs better, with a slight favor for MineRX especially in smaller datasets.

9.7. Runtime Experiments

In this section the runtimes of the various algorithms on the test datasets are given. The algorithms Mine, MineR and MineT (the ones without using vertex expiration when otherwise possible), are skipped, since on the test system they are equally fast than the ones with vertex expiration. In the following charts no difference could be examined for them (the difference is less than one percent in computation time). This can be explained when looking at the test systems hardware and the datasets under test, since plenty of memory is available. Therefore, the computed isochrones are too small to cause a big difference in calculation time. The results for the synthetic datasets is given in Figure 9.6. The x-axis corresponds to the maximal duration d_{max} in minutes, while on the y-axis the computation time (in seconds) is plotted.



(a) Grid (b) Spider

Figure 9.6.: Computation runtime in synthetic networks.

The MineRX (and MineRB) algorithm performs best in the synthetic networks. MineT(X) is able to keep up for grid layouts, but not in spider networks. Spider networks are optimal for circular ranges, since with only one loading operation all the vertices are loaded into memory. MineTX needs to use at least four tile ranges, since the center of the spider network is exactly where four tile ranges meet. MineX performs the worst throughout the synthetic networks, while MDijkstra performs bad for small durations d_{max}, but delivers the smallest computation times for large d_{max}.

The computation times for the real-world datasets is given in Figure 9.7.

The runtime of the isochrone computation is different across the various datasets. For small datasets the best performance is achieved when loading the whole network into memory and performing network expansion solely there. Only for small isochrones with a maximal duration of less than ten minutes

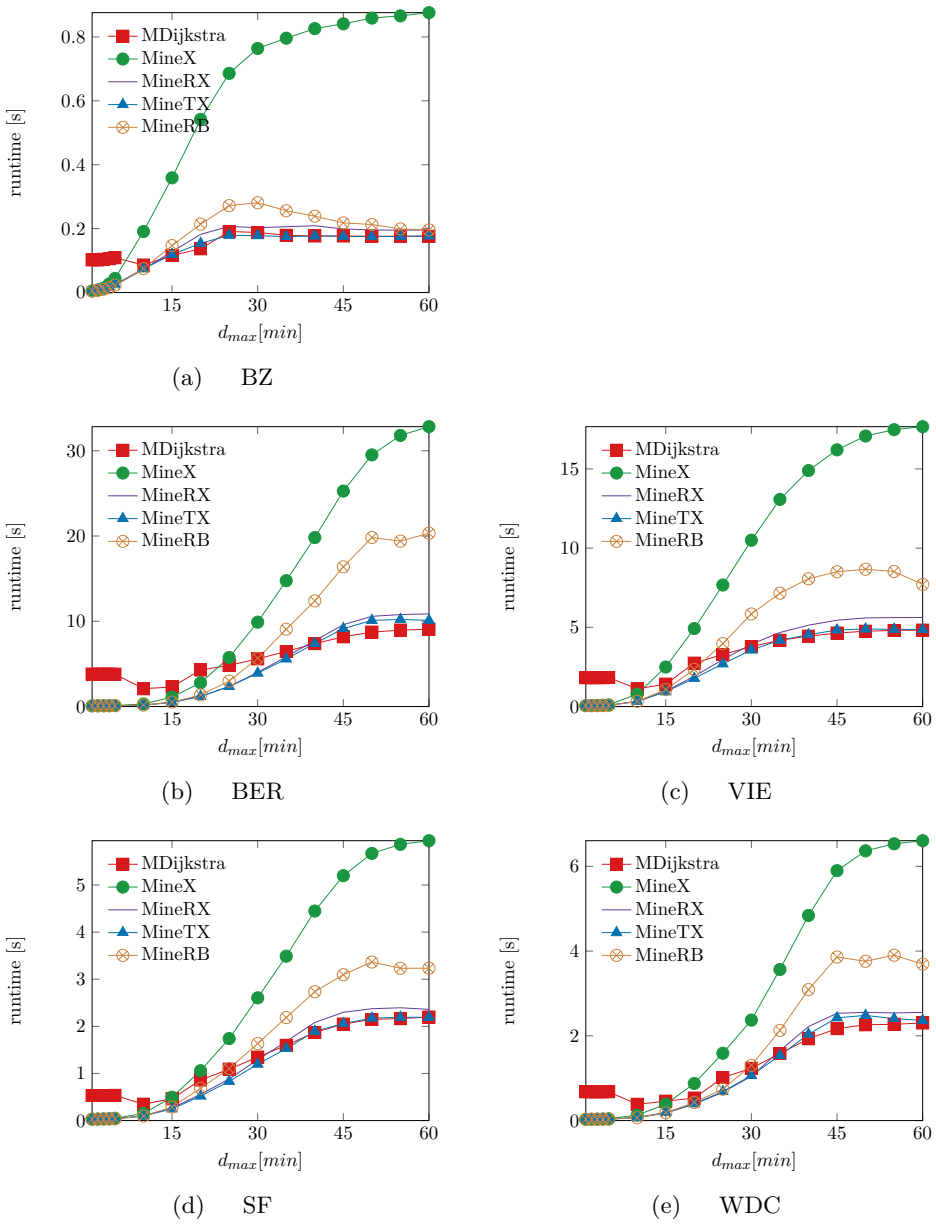


Figure 9.7.: Computation runtimes on real-world datasets.

other approaches perform better. For these very small isochrones MineRX performs best with MineX and MineTX performing nearly as good. As soon as the maximal duration d_{max} is increased, MineTX delivers favourable runtimes. For large isochrones in small datasets using MDijkstra provides the fastest computation, followed by MineTX that delivers the second best run-

times in front of MineRX, MineRB and MineX. A closer look is plotted for the datasets of San Francsico in Figure 9.8.

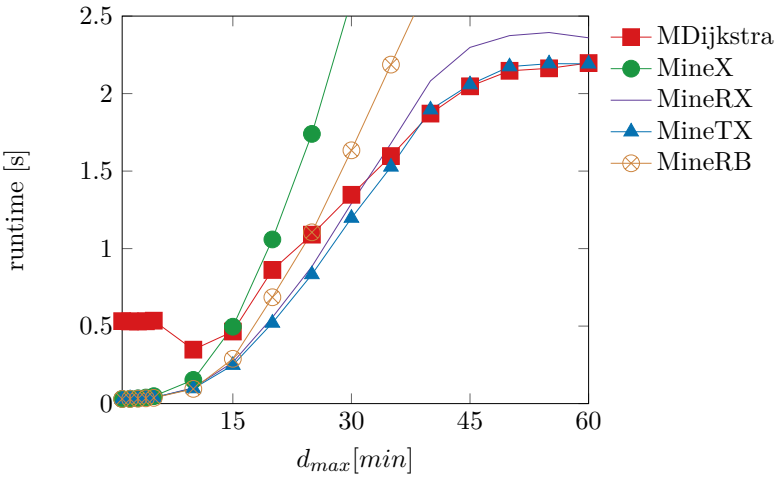


Figure 9.8.: Detailed computation runtime for the dataset of San Francisco.

It can be observed that MineTX performs better than MineRX with a bigger difference for maximal durations d_{max} above 30 minutes. For these durations, MineTX is able to stay closer to the runtimes of MDijkstra making it favorable over MineRX. Therefore, when only one algorithm should be used to create isochrones of arbitrary d_{max}, MineTX is the algorithm performing best.

9.8. Break-Even Points and Network Independence

For real-world datasets it has been seen that there is a point where MDijkstra becomes the fastest algorithm although it loads the entire network into memory. Therefore, a major question regarding loading data from a data source is, how long the computation stays faster when using a database. This can be determined by looking at so called break-even points that reflect the point of intersection in the Figure 9.7 from above. The results for the real-world datasets is given in Figure 9.9. The x-axis shows the dataset under test, while on the y-axis the d_{max} (in seconds) of the break-even point is plotted. Break-even points with MDijkstra should be as late as possible, therefore high values in the figure are better.

It can be seen that MineTX performs best, because the intersection point occurs the latest. The range batching optimization of MineRX (the algorithm MineRB) does not improve the original algorithm. Depending on the dataset MineTX performs about five percent better than MineRX (for the dataset of

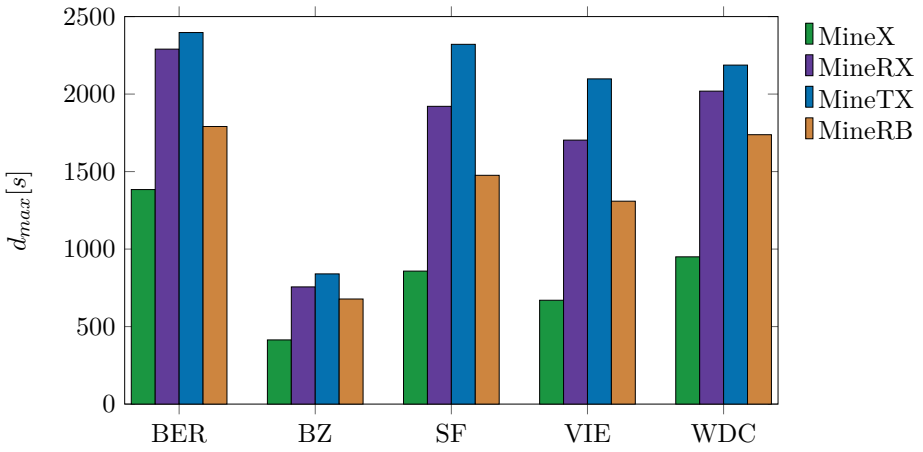


Figure 9.9.: Break-even points on real-world datasets.

Berlin) to a little more than twenty percent better (for the datasets of San Francisco and Vienna).

Since the results vary across the modeled cities the next evaluation checks the independence of the algorithms to the size of the datasets. This is done by computing a reachable area holding exactly 3000 vertices. Fixing the result size and not the maximal duration as done in Section 9.6 and 9.7 allows a comparison across the datasets. The result is plotted in Figure 9.10. The x-axis lists the dataset under test, while on the y-axis the computation time (in seconds) to create a reachable area including 3000 vertices is given. The computation time should not only be low for an algorithm, it should also be of the same size across the different datasets to indicate scalability.

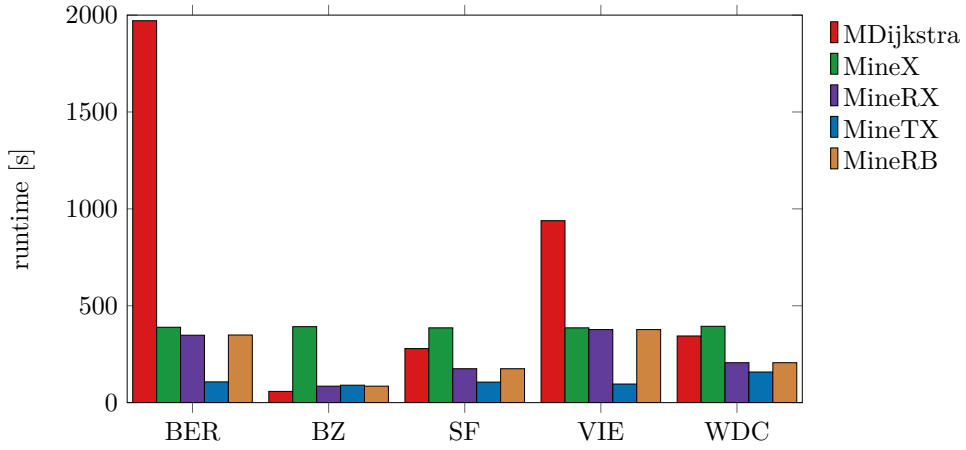


Figure 9.10.: Network independence of the algorithms on real-world datasets.

As expected MDijkstra is highly dependent on the network size. In Berlin and Vienna (the two largest datasets) its runtime is high. For small datasets, such as Bolzano, the runtime is very low. The cause for this behavior is the initial loading of the entire network. In contrast to the in-memory computation the approaches using incremental network expansion are not as dependent on the datasets size. MineX shows only minimal variations over all datasets. MineTX also stays constant across the datasets with slight variation in the dataset of Washington, D.C. (WDC). The runtimes of MineRX and MineRB also vary across the tested datasets, although in general runtimes are less than the ones of MineX. Since MineTX delivers the best runtimes (except for the smallest dataset representing Bolzano) and variation across the datasets is low, its overall performance outperforms previous approaches.

9.9. Elevation aware performance

The evaluations in the previous sections did not include elevation awareness to be comparable with previously carried out research. Differences in computation runtime or memory consumption for certain maximal durations d_{max} would occur when using elevation aware computing, but that would not necessarily indicate a change in performance. The reason for this is that elevation awareness also changes the reachable area. It is expected that performance increases if big parts of the reachable area include inclines for the same d_{max}. For declines the reachable area would get bigger so the performance would look bad when compared to the same d_{max} when not using elevation information. Also when using isochrones of the same size (referring to the number of edges and vertices included in the reachable area), like it has been done during evaluating the network independence of the algorithms in the previous section, comparability would not be guaranteed. The reason here is that although the size is the same, the isochrone would still be different. It could be that less continuous space and time edges are included that got replaced with discrete edges. Therefore more queries to schedules would occur causing problems regarding comparability.

Therefore, another approach was taken to benchmark elevation aware computation. All edges within a dataset are loaded that then are used to compute the reachable travel speeds on it. Then it is recorded how long the computation (with and without elevation) took per edge. To produce meaningful results all edges of the Bolzano datasets (16476 edges) were iterated multiple times (500.000 times). Then the time needed for one edge was calculated from the overall runtime. The results are given in Table 9.4.

Mode	Time per edge
Fixed (− elevation)	5.8 nanosec
Simple (− elevation)	11.06 nanosec
Cycling (− elevation)	15.55 nanosec
Cycling (+ elevation; − precomputation)	8216.96 nanosec
Cycling (+ elevation; + precomputation)	22.62 nanosec
Walking (− elevation)	10.89 nanosec
Walking (+ elevation; − precomputation)	8233.54 nanosec
Walking (+ elevation; + precomputation)	87.94 nanosec
User profile	22.46 nanosec

Table 9.4.: Runtime for elevation aware computation

It can be seen that the two modes "Cycling (+ elevation; − precomputation)" and "Walking (+ elevation; − precomputation)" need much more time than the other modes. These two modes compute the achievable speed on the edge by using all the hypervertices of an edge. The length for all edge segments is computed together with the difference in elevation. From these two values the speed on each segment is computed. The speed values of all segments are then averaged resulting in the achievable speed on the edge. The determination of the edge segments length is computational intense compared to the other modes, since spatial reference systems might have to be converted and the computation itself computes the orthodromic distance. Therefore, the precomputation explained in Section 6.3 that used average incline, average decline and the length on which the edge inclines (in percentage) is applied. With the help of these three values the same modes perform much better (around 100 times faster), while still regarding elevation. Without elevation awareness the computation for the walking mode is around eight times faster. It can also be seen that using user profiles or taking elevation into account when traveling by bike perform equally fast at around 22 nanoseconds for a single edge. Not regarding elevation takes about ten nanoseconds per edge to handle the maximum speed allowed on that edge together with the type and surface of the edge. For bicycles it takes slightly more time (15 nanoseconds), since more speed factors for different surfaces are known for that mode of traveling. Not taking any weights of the graphs edge into account is of course the fastest, but can only be applied when computing isodistances, not isochrones (isochrones always regard the maximum speed allowed).

If the results are seen in context with the sizes of the datasets, then it becomes clear that elevation awareness does not influence performance of the isochrone computation much. Even the largest dataset under test (Berlin) contains only 716396 edges. Even if it had one million edges, and a reachable area would be computed that covers the whole dataset, then the elevation would add a total

of 88msec when using the mode “Walking (+ elevation; + precomputation)”. In comparison to the overall runtime of the isochrone creation (that is around ten seconds for an isochrone with a maximal duration of one hour that still does not cover the whole dataset) this is below one percent.

CHAPTER 10

Conclusion

10.1. Summary

In this thesis improvements for isochrones in Multimodal Spatial Networks were made. On the one hand new algorithms capable of creating isochrones in such networks were introduced that outperform previous approaches. On the other hand enhancements to the field of application of isochrones were made. Those include the definition and calculation of averaged and time-invariant isochrones, as well as elevation aware computation. Re-visiting and improving data structures used by existing algorithms optimize all calculations. Memoization was introduced to the field of isochrones in Multimodal Spatial Networks, which further decreased computation time whenever previously known results are available.

With the establishment of a generalized dataset creation workflow the problem of lacking comparability across different methods and throughout various geographical locations was tackled. This and the newly developed application "IsoMap" ease the computation of isodistances and isochrones in unimodal and multimodal networks even for people not familiar with the topic. This application visually presents the calculation results in an appropriate way that can be interactively explored and analyzed. The dataset creation tool

"osmPti2mmds" allows the creation of the underlying datasets and is able to include information from various external data sources, e.g. to include information about elevation and schedules. Various settings that are of importance throughout the computation can be modified, so that further insights in the topic of isochrones in Multimodal Spatial Networks can be gained. For the elevation awareness various methods were introduced that target for different modalities. Besides walking also traveling by bicycle as well as methods respecting the surface and type of a street were implemented. Subsequently, a way to tailor isochrones to individuals with the help of so called "user profiles" was presented.

An empirical evaluation shows the memory consumption, the computation runtimes as well as the network independence of the algorithms proposed in this thesis. It outlines that batch loading of circular ranges does not lead to an improvement. Although memory consumption stays the same and less calls to the data source are needed, the problem of calculating (possibly large) regions of the network more than once adds to the calculations runtime. In contrast, using tile regions for loading the data from an underlying data source is of benefit for all the real-world datasets when compared to the previously available algorithms using incremental network expansion. This was not only shown in a publication, but was further improved with adaptive zooming. After carrying out a precomputation step that determines the optimal adaptive zoom ranges, the proposed method is able to compute large results in large networks within a feasible amount if time. In addition, most of the disadvantages of the previous approaches were addressed by a newly developed algorithm that was named "MineTX". As a result, MineTX delivers the best break-even points and better network independence than MineRX. When using the new algorithm computations times are low for isochrones with small maximal duration d_{max} and stay closer to the runtime of loading the whole network into memory (MDijkstra) for large isochrones.

All the datasets and the source-code of the algorithms, enhancements and tools that were created throughout this thesis are freely available under an open-source license. More information about this is available on the website `https://dbis-isochrone.uibk.ac.at` [60] as well as on `https://git.uibk.ac.at/dbis-isochrone` [61].

10.2. Future Research Directions

Open issues remain that have not been addressed throughout the thesis. The results of the elevation aware computations could be further improved by uti-

lizing more accurate data. The SRTM data that is used provides a resolution of 30 x 30 meters. Local datasets and newer approaches like OpenDEM [100] provide higher resolutions and would allow to create isochrones that model the real-world conditions better. Such approaches have not yet been implemented, since global coverage is missing and therefore comparisons across the globe would not be possible. If comparability is neglectable and accuracy is the major concern, using such DEMs could be beneficial.

The datasets that were used for the empirical evaluation were created with the help of boundary polygons that represent cities or specific regions. The created Multimodal Spatial Networks allow for isochrone computation, but only within one of these regions. Therefore, seamless isochrone computation across different datasets could be of interest. Not only computing across datasets, but also applying incremental updates to the datasets itself could ease data handling. This would also allow to include real-time data like information about traffic jams or delayed buses that arrive behind schedule. Although this includes handling of additional data sources, such as GTFS Realtime, it could add valuable information especially when traveling by car or when using public transport.

The incremental computation of isochrones that was introduced in Section 5.4 could be further extended. At the moment changing maximal durations use memoization. If the principle is applied to other parameters, like traveling speed or starting date and time, the calculation runtime of isochrones in Multimodal Spatial Networks could possibly be reduced even more.

Future algorithms able to create multimodal isochrones could focus on adopting other approaches from the field of routing, including Contraction Hierarchies (CH). Utilizing current hardware, like Solid-State-Drives, fast graphic cards or even accelerated processing units (APUs) that enable zero-copy techniques between the central processing unit (CPU) and the graphics processing unit (GPU) [89] offer the potential to decrease computation times. Performance improvements could address the performance of creating the line around the areas computed with the algorithm discussed throughout this thesis. This would speed-up visualizations within IsoMap and would also allow fast computation of averaged and time-invariant isochrones that rely on the geometric representation of the results rather than on the graph's edges and vertices.

The creation of datasets and user profiles is done with the help of command line scripts. Building sophisticated graphical user interfaces could ease these tasks and would allow usage even for people not that familiar with computer systems.

APPENDIX A

Appendix

A.1. Way Type Classes

The following Table A.1 lists the rules that are applied to classify edges within road network graph into way type classes in order to create and use profiles explained in Section 6.4. The rules themselves have been defined together with M. Malfertheiner [70].

ID	Class	Rule
0	MOTORWAY	$highway \in \{motorway, motorway\_link, trunk, trunk\_link\}$
1	ROAD	$highway \in \{primary, primary\_link, secondary, secondary\_link\}$
2	TERTIARY_ROAD	$highway \in \{tertiary, tertiary\_link\}$
3	UNCLASSIFIED_PAVED	$highway \in \{unclassified\} \wedge \neg UNCLASSIFIED\_UNPAVED$
4	UNCLASSIFIED_ UNPAVED	$highway \in \{unclassified\} \wedge (surface \in unpaved\_surface\_set \vee (tracktype \neq null \wedge tracktype \neq grade1))$

5	SMALL_WAY_PAVED	$highway \in \{residential, living\_street, service\} \wedge \neg SMALL\_WAY\_UNPAVED$
6	SMALL_WAY_UNPAVED	$highway \in \{residential, living\_street, service\} \wedge (surface \in unpaved\_surface\_set \vee (tracktype \neq null \wedge tracktype \neq grade1))$
7	TRACK_EASY	$highway \in \{track\} \wedge \neg(TRACK\_MIDDLE \vee TRACK\_HARD)$
8	TRACK_MIDDLE	$highway \in \{track\} \wedge \neg TRACK\_HARD \wedge tracktype \in \{grade2, grade3\} \wedge surface \notin paved\_surface\_set$
9	TRACK_HARD	$highway \in \{track\} \wedge tracktype \in \{grade4, grade5\} \wedge surface \notin paved\_surface\_set$
10	PATH_EASY	$highway \in \{path\} \wedge \neg(PATH\_MIDDLE \vee PATH\_HARD)$
11	PATH_MIDDLE	$highway \in \{path\} \wedge \neg PATH\_HARD \wedge (smoothness \in \{bad, very\_bad\} \vee sac\_scale \in \{hiking\} \vee mtb : scale \in \{1, 2, 3\} \wedge surface \notin paved\_surface\_se \wedge \neg bicycle\_intended)$
12	PATH_HARD	$highway \in \{path\} \wedge (smoothness \in \{horrible, very\_horrible\} \vee sac\_scale \in \{demanding\_mountain\_hiking, mountain\_hiking\} \vee mtb : scale \in \{4, 5\}$
13	CYCLEWAY	$(type = bicycle \wedge network \in \{icn, ncn, rcn, lcn\}) \vee highway = cycleway \vee bicycle = designated$
14	MTB_CYCLEWAY	$type = mtb \vee (bicycle = designated \wedge surface \notin paved\_surface\_set)$
15	PUSHING_SECTION	$highway \in \{footway, pedestrian, steps\} \vee surface = ice$

Table A.1.: Rules to classify edges into way type classes

List of Algorithms

List of Figures

List of Tables

Bibliography

[1] A. M. Andrew. "Another Efficient Algorithm for Convex Hulls in Two Dimensions". In: *Information Processing Letters* 9.5 (1979), pp. 216–219.

[2] A. Antrim, S. J. Barbeau, et al. "The Many Uses of GTFS Data – Opening the Door to Transit and Multimodal Applications". In: *Location-Aware Information Systems Laboratory at the University of South Florida* (2013), p. 4.

[3] C. Arias Muñoz, S. Corti, M. Molinari, D. Oxoli, and G. Prestifilippo. "City Focus: A web-based interactive 2D and 3D GIS application to find the best place in a city, using open data and open source software". In: *PeerJ Preprints* 4 (2016).

[4] Azavea. *GeoTrellis Transit*. URL: `http://transit.geotrellis.com` (visited on 07/2017).

[5] J. Baez. *Network Theory*. URL: `http://math.ucr.edu/home/baez/econ.pdf` (visited on 08/2017).

[6] V. K. Balakrishnan. *Graph Theory*. McGraw-Hill, 1997.

[7] M. Barclay and A. Galton. "Comparison of region approximation techniques based on Delaunay Triangulations and Voronoi Diagrams". In: *Computers, Environment and Urban Systems* 32.4 (2008), pp. 261–267.

[8] G. V. Batz, D. Delling, P. Sanders, and C. Vetter. "Time-dependent Contraction Hierarchies". In: *Proceedings of 11th Workshop on Algorithm Engineering and Experiments (ALENEX)*. SIAM, 2009, pp. 97–105.

[9] V. Bauer, J. Gamper, R. Loperfido, S. Profanter, S. Putzer, and I. Timko. "Computing Isochrones in Multi-Modal, Schedule-Based Transport Networks". In: *Proceedings of the 16th annual ACM International Symposium on Advances in Geographic Information Systems (SIGSPATIAL)*. GIS '08. ACM. Irvine, California, 2008, 78:1–78:2.

[10] M. Baum, V. Buchhold, J. Dibbelt, and D. Wagner. "Fast Exact Computation of Isochrones in Road Networks". In: *Proceedings of the 15th International Symposium on Experimental Algorithms (SEA)*. Springer, 2016, pp. 17–32.

[11] R. Bierbauer. "Comparing routing implementations in various database systems". Master Thesis. University of Innsbruck, Sept. 2016.

[12] N. Biggs, E. K. Lloyd, and R. J. Wilson. *Graph Theory, 1736-1936*. Oxford University Press, 1976.

[13] *Bing Maps Isochrone Demo*. URL: `http://bmlabs.azurewebsites.net/v8-Isochrone` (visited on 07/2017).

[14] J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications*. Macmillan Press, 1976.

[15] M. A. Brovelli, M. Minghini, M. Molinari, and P. Mooney. "Towards an Automated Comparison of OpenStreetMap with Authoritative Road Datasets". In: *Transactions in GIS* (2016).

[16] Z. Chen, Y. Liu, R. C.-W. Wong, J. Xiong, G. Mai, and C. Long. "Efficient Algorithms for Optimal Location Queries in Road Networks". In: *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*. ACM. 2014, pp. 123–134.

[17] Conveyal. *Conveyal Analyst*. URL: `http://conveyal.com/projects/analyst` (visited on 07/2017).

[18] Conveyal. *Conveyal Analyst on Github*. URL: `https://github.com/conveyal/analyst-server` (visited on 07/2017).

[19] T. H. Cormen. *Introduction to Algorithms*. MIT press, 2009.

[20] Crunchy Bagel. *Transitfeeds.com*. URL: `https://transitfeeds.com` (visited on 10/2017).

[21] DB Vertrieb GmbH. *Umkreissuchder der Deutschen Bahn*. URL: `https://www.bahn.de/regional/view/erlebnis/umkreissuche.shtml` (visited on 07/2017).

[22] D. Delling. "Time-Dependent SHARC-Routing". In: *Algorithmica* 60.1 (2011), pp. 60–94.

[23] E. W. Dijkstra. "A Note on Two Problems in Connexion with Graphs". In: *Numerische Mathematik* 1.1 (1959), pp. 269–271.

[24] D. P. Doane and L. E. Seward. “Measuring Skewness: A Forgotten Statistic”. In: *Journal of Statistics Education* 19.2 (2011), pp. 1–18.

[25] Y. Du, D. Zhang, and T. Xia. “The Optimal-Location Query”. In: *Proceedings of the 9th International Symposium on Advances in Spatial and Temporal Databases (SSTD)*. Vol. 2005. Springer. 2005, pp. 163–180.

[26] M. Duckham, L. Kulik, M. Worboys, and A. Galton. “Efficient generation of simple polygons for characterizing the shape of a set of points in the plane”. In: *Pattern Recognition* 41.10 (2008), pp. 3224–3236.

[27] D. Dzafic, S. Klug, D. Franke, and S. Kowalewski. “Routing über Flächen mit SpiderWebGraph”. In: 1 (2015), pp. 516–525.

[28] H. Edelsbrunner. *Weighted Alpha Shapes*. University of Illinois at Urbana-Champaign, Department of Computer Science, 1992.

[29] H. Edelsbrunner, D. Kirkpatrick, and R. Seidel. “On the Shape of a Set of Points in the Plane”. In: *IEEE Transactions on information theory* 29.4 (1983), pp. 551–559.

[30] Factbook, CIA. *The world factbook*. 2017. URL: `https://www.cia%20gov/library/publications/the-world-factbook` (visited on 10/2017).

[31] P. Frankl. “What must be contained in every oriented k-uniform hypergraph”. In: *Discrete Mathematics* 62.3 (1986), pp. 311–313.

[32] M. L. Fredman and R. E. Tarjan. “Fibonacci Heaps and their Uses in Improved Network Optimization Algorithms”. In: *Journal of the ACM (JACM)* 34.3 (1987), pp. 596–615.

[33] J. Gamper, M. Böhlen, W. Cometti, and M. Innerebner. “Defining Isochrones in Multimodal Spatial Networks”. In: *Proceedings of the 20th ACM International Conference on Information and Knowledge Management (CIKM)*. ACM. 2011, pp. 2381–2384.

[34] J. Gamper, M. Böhlen, and M. Innerebner. “Scalable Computation of Isochrones with Network Expiration”. In: *Proceedings of 24th International Conference on Scientific and Statistical Database Management (SSDBM)*. ACM. Springer, 2012, pp. 526–543.

[35] R. Garcia, J. P. de Castro, E. Verdu, M. J. Verdu, and L. M. Regueras. “Web Map Tile Services for Spatial Data Infrastructures: Management and Optimization”. In: *Cartography – A Tool for Spatial Analysis*. InTech, 2012.

[36] Geofabrik GmbH. *Geofabrik*. URL: `https://www.geofabrik.de` (visited on 10/2017).

[37] Geomatics and Earth Observation laboratory (GEOlab) of Politecnico di Milano. *CityFocus*. URL: `http://muvias.eoapps.eu/cityfocus` (visited on 07/2017).

[38] J.-F. Girres and G. Touya. "Quality Assessment of the French OpenStreetMap Dataset". In: *Transactions in GIS* 14.4 (2010), pp. 435–459.

[39] M. F. Goodchild. "Citizens as Sensors: The World of Volunteered Geography". In: *GeoJournal* 69.4 (2007), pp. 211–221.

[40] Google. *Google Inside Search: The Knowledge Graph*. URL: `https://www.google.com/intl/bn/insidesearch/features/search/knowledge.html` (visited on 10/2017).

[41] R. L. Graham. "An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set". In: *Information Processing Letters* 1.4 (1972), pp. 132–133.

[42] A. Graser. Feb. 2011. URL: `https://anitagraser.com/2011/02/09/creating-catchment-areas-with-pgrouting-and-qgis` (visited on 07/2017).

[43] A. Graser. July 2013. URL: `https://anitagraser.com/2013/07/07/public-transport-isochrones-with-pgrouting/` (visited on 07/2017).

[44] HaCon Ingenieurgesellschaft mbH. *HaCon Umkreissuche*. URL: `http://www.hacon.de/unternehmen/presse/pressearchiv/pressearchiv-2012/deutsche-bahn-setzt-auf-die-umkreissuche-von-hacon` (visited on 07/2017).

[45] J. Han and C. Moraga. "The Influence of the Sigmoid Function Parameters on the Speed of Backpropagation Learning". In: *Proceedings of the International Workshop on Artificial Neural Networks (IWANN): From Natural to Artificial Neural Computation, Malaga-Torremolinos, Spain, June 7-9*. Springer-Verlag. 1995, pp. 195–201.

[46] P. E. Hart, N. J. Nilsson, and B. Raphael. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (July 1968), pp. 100–107.

[47] L. S. Heath and A. A. Sioson. "Multimodal Networks: Structure and Operations". In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 6.2 (Apr. 2009), pp. 321–332.

[48] O. Huisman and A. d. B. Rolf. "Principles of geographic information systems". In: ().

[49] iGeolise. *TravelTime Platform*. URL: `http://www.traveltimeplatform.com` (visited on 07/2017).

[50] *iGeolise on Github*. URL: `https://github.com/igeolise` (visited on 07/2017).

[51] M. Innerebner. "Isochrones in Multimodal Spatial Networks". PhD Thesis. Free University of Bozen-Bolzano, 2013.

[52] M. Innerebner, M. Böhlen, and J. Gamper. "ISOGA: A System for Geographical Reachability Analysis". In: *Proceedings of the 12th International Symposium on Web and Wireless Geographical Information Systems (W2GIS), Banff, AB, Canada, April 4-5*. Berlin, Heidelberg: Springer, 2013, pp. 180–189.

[53] R. A. Jarvis. "On the Identification of the Convex Hull of a Finite Set of Points in the Plane". In: *Information Processing Letters* 2.1 (1973), pp. 18–21.

[54] R. E. Kalman et al. "A New Approach to Linear Filtering and Prediction Problems". In: *Journal of basic Engineering* 82.1 (1960), pp. 35–45.

[55] S. Kaufmann. "Opening Public Transit Data in Germany". Master Thesis. University of Ulm, 2014.

[56] O. Kounadi. "Assessing the Quality of OpenStreetMap Data". Master Thesis. University College of London, 2009.

[57] N. Krismer. "INNsochrone". In: (2015).

[58] N. Krismer. "Interaktive Karten und HTTP/2". In: (July 2016).

[59] N. Krismer. *Datasets modeling Multimodal Spatial Networks*. URL: `https://dbis-owncloud.uibk.ac.at/index.php/s/kgjm3CItQJ6P374` (visited on 10/2017).

[60] N. Krismer. *DBIS-Isochrone Homepage and the IsoMap application*. URL: `https://dbis-isochrone.uibk.ac.at` (visited on 10/2017).

[61] N. Krismer. *Version Control System for Isochrone Source Code*. URL: `https://git.uibk.ac.at/dbis-isochrone` (visited on 09/2017).

[62] N. Krismer, J. Gamper, and G. Specht. "Isochrones in Multimodal Spatial Networks". In: *AGIT Postersession* (July 2014).

[63] N. Krismer, J. Gamper, and G. Specht. "Reachability Calculation based on Average Isochrones Regarding Time Distribution". In: *AGIT Postersession* (July 2015).

[64] N. Krismer, D. Silbernagl, J. Gamper, and G. Specht. "osmPti2mmds - Erstellung von multimodalen Datensets aus OpenStreetMap und ÖPNV-Informationen". In: *AGIT Journal* 2 (2016), pp. 185–190.

[65] N. Krismer, D. Silbernagl, J. Gamper, and G. Specht. "Computing Isochrones in Multimodal Spatial Networks Using Tile Regions". In: *Proceedings of the 29th International Conference on Scientific and Statistical Database Management (SSDBM), Chicago, Illinois, USA, June 27 - 29*. ACM, 2017, 33:1–33:6.

[66] N. Krismer, D. Silbernagl, M. Malfertheiner, and G. Specht. "Elevation Enabled Bicycle Router Supporting User-Profiles". In: *Proceedings of the 28th GI-Workshop Grundlagen von Datenbanken (GvDB), Nörten Hardenberg, Germany, May 24 - 27*. 2016, pp. 74–79.

[67] N. Krismer, G. Specht, and J. Gamper. "Incremental Calculation of Isochrones Regarding Duration". In: *Proceedings of the 26th GI-Workshop Grundlagen von Datenbanken (GvDB), Bozen-Bolzano, Italy, October 21-24*. 2014, pp. 41–46.

[68] K-SOL S.r.l. *iso4app*. URL: `https://www.iso4app.net` (visited on 07/2017).

[69] Z. Lin and Y. Li. "An Efficient Algorithm for Intersection, Union and Difference between Two Polygons". In: *Proceedings of the International Conference on Computational Intelligence and Software Engineering (CISE)*. IEEE. 2009, pp. 1–4.

[70] M. Malfertheiner. "feveR - A prototypical implementation of a profile and elevation aware bicycle router". Master Thesis. University of Innsbruck, June 2016.

[71] Mapbox. *Mapbox data*. URL: `https://www.mapbox.com/data-platform/country/` (visited on 10/2017).

[72] Mapzen. *Transit.land*. URL: `https://transit.land` (visited on 10/2017).

[73] S. Marciuska and J. Gamper. "Determining Objects within Isochrones in Spatial Network Databases". In: *East European Conference on Advances in Databases and Information Systems (ADBIS)*. Springer. 2010, pp. 392–405.

[74] J. Masó, K. Pomakis, and N. Julià. *Web Map Tile Service Implementation Standard*. Tech. rep. OGC 07-057r7. Open Geospatial Consortium, 2010.

[75] B. McHugh. "Pioneering Open Data Standards: The GTFS Story". In: *Beyond Transparency: Open Data and the Future of Civic Innovation* (2013), pp. 125–135.

[76] K. Mehlhorn and P. Sanders. *Algorithms and Data Structures: The Basic Toolbox*. Springer Science & Business Media, 2008.

[77] M. Melkemi and M. Djebali. "Computing the Shape of a Planar Points Set". In: *Pattern Recognition* 33.9 (2000), pp. 1423–1436.

[78] D. Michie. "Memo Functions and Machine Learning". In: *Nature* 218.5136 (1968), pp. 19–22.

[79] C. Moeller. *Osm2po*. URL: `https://osm2po.de/` (visited on 10/2010).

[80] P. Mooney, P. Corcoran, and A. C. Winstanley. "Towards Quality Metrics for OpenStreetMap". In: *Proceedings of the 18th annual ACM International Symposium on Advances in Geographic Information Systems (SIGSPATIAL)*. ACM. 2010, pp. 514–517.

[81] A. Moreira and M. Y. Santos. "Concave Hull: A K-Nearest Neighbours Approach for the Computation of the Region Occupied by a Set of Points". In: *Proceedings of the 2nd International Conference on Computer Graphics Theory and Applications (GRAPP), Barcelona, Spain, 8-11 March, 2007*. Institute for Systems, Technologies of Information, Control, and Communication(INSTICC) Press, 2007.

[82] Motion Intelligence. *Route360*. URL: `https://www.route360.net` (visited on 07/2017).

[83] MySociety. *Mapumental*. URL: `https://mapumental.com` (visited on 07/2017).

[84] Naturtrip GmbH. *Naturtrip.org*. URL: `https://naturtrip.org` (visited on 07/2017).

[85] P. Neis. "Location Based Services mit OpenStreetMap Daten". Master Thesis. Fachhochschule Mainz, 2008.

[86] P. Neis and D. Zielstra. "Recent Developments and Future Trends in Volunteered Geographic Information Research: The Case of OpenStreetMap". In: *Future Internet* 1 (2014), pp. 76–106.

[87] P. Neis and A. Zipf. "OpenRouteService.org is three times "Open": Combining OpenSource, OpenLS and OpenStreetMaps". In: *Proceedings of the GIS Research UK 16th Annual Conference (GISRUK 08), Manchester*. 2008.

[88] M. Neteler, M. H. Bowman, M. Landa, and M. Metz. "GRASS GIS: a multi-purpose Open Source GIS". In: *Environmental Modelling & Software* 31 (2012), pp. 124–130.

[89] K. Nilakant and E. Yoneki. "On the Efficacy of APUs for Heterogeneous Graph Computation". In: *Proceedings of the 4th Workshop on Systems for Future Multicore Architectures (SFMA), Amsterdam, The Netherlands, April 13th*. 2014.

[90] J. Nishimura. "The connectivity of graphs of graphs with self-loops and a given degree sequence". In: *arXiv preprint arXiv:1701.04888* (2017).

[91] Open Geospatial Consortium (OGC). *OpenGIS Simple Features Implementation Specification for SQL, Revision 1.1*. URL: `http://www.opengeospatial.org/standards/sfs` (visited on 09/2017).

[92] Open Source Geospatial Foundation (OSGeo). *Workshop on PostGIS - Section 8: Spatial Indexing*. URL: http://revenant.ca/www/postgis/workshop/indexing.html (visited on 09/2017).

[93] OpenStreetMap contributors. *Isochrone - OpenStreetMap Wiki*. URL: http://wiki.openstreetmap.org/wiki/Isochrone (visited on 07/2017).

[94] OpenStreetMap contributors. *Slippy map tilenames - OpenStreetMap Wiki*. URL: https://wiki.openstreetmap.org/wiki/Slippy%5C_map%5C_tilenames (visited on 07/2017).

[95] OpenStreetMap contributors. *Transiki*. URL: http://wiki.openstreetmap.org/wiki/Transiki (visited on 10/2017).

[96] OpenStreetMap Fonudation (OSMF). *OpenStreetMap*. URL: https://www.openstreetmap.org (visited on 10/2017).

[97] *OpenTripPlanner*. URL: http://www.opentripplanner.org (visited on 07/2017).

[98] *OpenTripPlanner on Github*. URL: https://github.com/opentripplanner/OpenTripPlanner (visited on 07/2017).

[99] Österreichische Länder bzw. Ämter der Landesregierung. *Graphenintegrations-Plattform GIP*. URL: http://www.gip.gv.at (visited on 07/2017).

[100] M. Over. *OpenDEM*. URL: http://www.opendem.info (visited on 10/2017).

[101] S. Paganotti. *Generate an isochrone map using Google Maps Api*. URL: http://sandropaganotti.com/generate-an-isochrone-map-using-google-maps-api (visited on 07/2017).

[102] D. A. Papa and I. L. Markov. "Hypergraph Partitioning and Clustering". In: *Handbook of Approximation Algorithms and Metaheuristics*. 2007, pp. 61–1.

[103] pgRouting Community. *pgRouting*. URL: http://pgrouting.org (visited on 10/2017).

[104] PostGIS Project Steering Committee (PSC). *PostGIS*. URL: https://postgis.net (visited on 10/2017).

[105] *Proceedings of the 28th GI-Workshop Grundlagen von Datenbanken (GvDB), Nörten Hardenberg, Germany, May 24 - 27*. 2016.

[106] E. Pyrga, F. Schulz, D. Wagner, and C. D. Zaroliagis. "Efficient Models for Timetable Information in Public Transportation Systems". In: *ACM Journal of Experimental Algorithmics* 12 (2007).

[107] U. Rainer. "Evaluation of spatial database queries regarding their performance". Master Thesis. University of Innsbruck, Dec. 2016.

[108] Rome2Rio Labs. *Rome2Rio*. URL: `https://www.rome2rio.com` (visited on 07/2017).

[109] S. Schröder, P. Karich, and M. Zilske. *Graphhopper*. URL: `https://www.graphhopper.com` (visited on 07/2017).

[110] D. Schultes. "Route Planning in Road Networks". In: *Science* 316.5824 (2007), p. 566.

[111] K. Schwarz. *The Archive of Interesting Code*. URL: `http://www.keithschwarz.com/interesting` (visited on 09/2017).

[112] C. E. Shannon. "Communication in the Presence of Noise". In: *Proceedings of the Institute of Radio Engineers (IRE)* 37.1 (Jan. 1949), pp. 10–21.

[113] D. Silbernagl, N. Krismer, N. Augsten, and G. Specht. "Recommending OSM Tags To Improve Metadata Quality". In: ACM, 2017.

[114] D. Silbernagl, N. Krismer, M. Malfertheiner, and G. Specht. "Optimization of Digital Elevation Models for Routing". In: *Proceedings of the 28th GI-Workshop Grundlagen von Datenbanken (GvDB), Nörten Hardenberg, Germany, May 24 - 27*. 2016, pp. 103–108.

[115] D. Silbernagl, N. Krismer, and G. Specht. "Comparing OSM Area-Boundary Data to DBpedia". In: *Proceedings of the 12th International Symposium on Open Collaboration, OpenSym 2016, Berlin, Germany, August 17 - 19*. 2016, 11:1–11:4.

[116] D. Silbernagl, N. Krismer, and G. Specht. "osmpg2java - Konvertierung von OSM-Datenbankelementen zu JTS-Objekten". In: *AGIT Journal* 2 (2016), pp. 179–184.

[117] Società Autobus Servizi d'Area (SASA) AG. *SASAbus: Open Data & APIs*. URL: `http://sasabus.org/opendata` (visited on 10/2017).

[118] K. Stolze. "SQL/MM Spatial – The Standard to Manage Spatial Data in a Relational Database System". In: *Proceedings of Datenbanksysteme in Business, Technologie und Web (BTW)*. 2003, pp. 247–264.

[119] The PostgreSQL Global Development Group. *PostgreSQL*. URL: `https://www.postgresql.org` (visited on 10/2017).

[120] S. Tischer and M. Mailer. "NaWo-a Tool for More Sustainable Residential Location Choice". In: *Transportation Research Procedia* 19 (2016), pp. 109–118.

[121] J. Topf. *TagInfo*. URL: `https://taginfo.openstreetmap.org/` (visited on 10/2017).

[122] R. J. Trudeau. *Introduction to Graph Theory*. Courier Corporation, 2013.

[123] Universität Innsbruck and Münchner Verkehrs- und Tarifverbund GmbH and Research Studios Austria Forschungsgesellschaft mbH. *nawo - Nachhaltige Wohnstandortentscheidungen*. URL: `http://www.wowohnen.eu` (visited on 07/2017).

[124] Verkehrsauskunft Österreich. *Verkehrsauskunft Österreich (VAO)*. URL: `https://www.verkehrsauskunft.at` (visited on 07/2017).

[125] Walk Score. *Professional Travel Time API*. URL: `https://www.walkscore.com/professional/travel-time-api.php` (visited on 07/2017).

[126] S. Wehrmeyer. *Mapnificent*. URL: `http://www.mapnificent.net` (visited on 07/2017).

[127] D. B. West. *Introduction to Graph Theory*. Vol. 2. Prentice hall Upper Saddle River, 2001.

[128] R. J. Wilson. *Introduction to Graph Theory*. 1972.

[129] J. Winder. "Echtzeit-Visualisierung von interaktiven Landkarten". Bachelor Thesis. University of Innsbruck.

[130] WNYC Data News Team. *Travel Time NYC*. URL: `https://project.wnyc.org/transit-time` (visited on 07/2017).

[131] X. Xiao, B. Yao, and F. Li. "Optimal location queries in road network databases". In: *Proceedings of the 27th IEEE International Conference on Data Engineering (ICDE)*. IEEE. 2011, pp. 804–815.

[132] W. YiHong and L. YongJiang. "An Isoline Generating Algorithm based on Delaunay". In: *Proceedings of the 2nd International Conference on Computer Engineering and Technology (ICCET)*. Vol. 7. IEEE. 2010, pp. V7-173–V7-176.