Computing Isochrones in Multimodal Spatial Networks using Tile Regions

Nikolaus Krismer
University of Innsbruck
Department of Computer Science
Innsbruck, Austria 6020
nikolaus.krismer@uibk.ac.at

Doris Silbernagl
University of Innsbruck
Department of Computer Science
Innsbruck, Austria 6020
doris.silbernagl@uibk.ac.at

Günther Specht
University of Innsbruck
Department of Computer Science
Innsbruck, Austria 6020
guenther.specht@uibk.ac.at

Johann Gamper
Free University of Bozen-Bolzano
Faculty of Computer Science
Bolzano, Italy 39100
gamper@inf.unibz.it

ABSTRACT

This paper describes a new method to compute isochrones in multimodal spatial networks, which aims at finding a good trade-off between memory usage and runtime. In the past, approaches based on Dijkstra's algorithm have been proposed. For small networks, the entire network is first loaded in main memory, where the network is expanded to determine the isochrone. For large networks that do not fit in main memory, approaches that load the network vertex-by-vertex during the expansion phase have been proposed. They keep the memory footprint minimal, but have to query the database for each node in the isochrone, which can be very time consuming. The method presented in this paper uses tiles (which are well known from interactive online maps) to realize chunk-loading of vertices by utilizing so-called tile regions. This approach significantly reduces the number of database requests, while keeping the memory usage low. Our method is able to compute isochrones even in large networks at a reasonable time. An experimental evaluation shows that the new algorithm clearly outperforms previous competitive approaches such as MINE and MINEX.

CCS CONCEPTS

• **Information systems →Geographic information systems;**

KEYWORDS

algorithms, isochrone, spatial database

ACM Reference format:
Nikolaus Krismer, Doris Silbernagl, Günther Specht, and Johann Gamper. 2017. Computing Isochrones in Multimodal Spatial Networks using Tile Regions. In *Proceedings of SSDBM '17, Chicago, IL, USA, June 27-29, 2017,* 6 pages.
DOI: http://dx.doi.org/10.1145/3085504.3085538

1 INTRODUCTION

Isochrones became more and more popular over the last few years. They have been integrated into several online services, such as OpenRouteService [7], Mapnificent [20] or Graphhopper [19]. In general, isochrones are a tool to carry out reachability analyses. They can be used for various purposes, such as urban planning, public transportation system optimization or to find places to spend some spare time nearby. The latter application has been implemented using isochrones for the city of Berlin by naturtrip.org [21].

Isochrones in multimodal networks are computed in a Dijkstra like fashion. Starting from the query point, the network is incrementally expanded in all directions until the maximum duration is reached. There are essentially two different strategies: Algorithms either perform the computation in main memory after loading the entire network, or they load the network incrementally one vertex at a time during the computation of the isochrone. The former approach incurs high memory costs, whereas the latter one requires many disk accesses and hence has a high query time. The algorithm proposed in this paper adopts an approach in between. Instead of loading one vertex at a time, it loads a small region of the network and performs then network expansion in memory. Whenever the expansion exceeds the regions already loaded in memory, a new network region is loaded. Although unimodal isochrones can be computed this way as well, the rest of this paper will focus on isochrones in multimodal spatial networks, which can be classified as continuous or discrete along, respectively, the space and the time dimension. Continuous space means that all points on an edge are accessible,

This research was funded with the help of the scholarship "Doktoratsstipendium aus der Nachwuchsförderung" of the University of Innsbruck which is held by the first author.

SSDBM '17, Chicago, IL, USA

DOI: http://dx.doi.org/10.1145/3085504.3085538

whereas in a discrete space network only the vertices can be accessed. Continuous time networks can be traversed at any point in time, discrete networks follow an associated schedule. For instance, the pedestrian network and the street network are continuous in time and space, whereas the public transport systems are discrete in both dimensions [8].

Throughout this paper a newly developed approach is presented that can be used to compute isochrones. It loads data in chunks from the spatial database holding the geographical information. The chunks used can be represented by so called tiles which are already known in context of online maps within the community of geoinformatics.

The rest of this paper is organized as follows: Section 2 discusses related work. Section 3 describes the newly developed approach. Section 4 presents the results of an empirical evaluation. Section 5 concludes the paper and outlines possible further research directions.

2 RELATED WORK

Isochrones in multimodal spatial networks have been defined by Gamper et al. in [8]. Algorithms used for isochrone computation, such as Dijkstra's algorithm [6], work well as long as the networks are small, but suffer when the network is large. An implementation of the Dijkstra algorithm that is able to work in a multimodal environment is presented by Bauer et al. in [3]. However, especially multimodal spatial networks tend to be large enough so that computation time can be reduced by taking care of the amount of data loaded. The approach described in [3] simply loads the whole network in main memory, making it unusable for large networks that do not fit in memory. An approach called Multimodal Incremental Network Expansion (MINE) was developed by Gamper et al. in [8] that loads data only when it is needed. The algorithm is still limited in terms of memory consumption when large isochrones are computed, but it is usable with networks of any size. It was later optimized to free memory as soon as possible, resulting in an approach called vertex expiration that was first implemented in an algorithm called Multimodal Incremental Network Expansion using vertex eXpiration (MINEX) [9].

Vertex expiration allows the computation of large isochrones since only a portion of the result is kept in memory that is needed for the correct functioning of the algorithm. According to the authors the runtime of MINE is superior to the runtime of the multimodal Dijkstra algorithm presented by Bauer et al. in [3] (called MDijkstra from now on) when computing small isochrones, but becomes slower for large isochrones. This also holds true for MINEX, although vertex expiration further decreases computation time. In general, there is a break-even point for each network that determines how long MINEX is faster than MDijkstra [9].

Approaches regarding isochrones and network partitioning have been presented by Baum et al. in [4]. However, algorithms using partitioning have not yet been applied to multimodal spatial networks.

The data forming a multimodal spatial network can be extracted from various online sources. A possible workflow for creating a multimodal spatial dataset suitable for isochrone calculation has been published in [12]. It uses two different sources: one to gather information about the continuous pedestrian network and one to acquire information about the discrete public transportation system.

The source from which the geospatial data is acquired is OpenStreetMap [18] (OSM). Quality analyses have been a topic of research since the beginnings of the project [11]. Depending on the geographic region it varies [10, 15], 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 [16] and more recently by Brovelli et al. in [5]. Especially in urban areas data is of good quality, whereas it can be poor in rural areas. The datasets used for evaluation will therefore represent cities. They are chosen to be of different size to show the behavior of the algorithms regarding network size.

The information about the public transportation systems is provided from the operators themselves. It is available in a format called General Transit Feed Specification (GTFS), which was presented by Google in 2006. It nowadays is the de facto standard for representing scheduled transit data and used by many different types of applications [1]. Other common formats in the context of such data, like the data model of the "Verband Deutscher Verkehrsunternehmen (VDV)" in some countries around Europe, can also be used if they are converted to GTFS first. All these information is either directly supplied at the operators web site or can be downloaded from various data exchange portals, like transit.land [13] or transitfeeds [2].

3 MULTIMODAL INCREMENTAL NETWORK EXPANSION USING TILE REGIONS

As described in Section 2, MINE and MINEX only load network parts that are really needed during expansion. Therefore, the two algorithms are optimal in terms of memory usage, but they are not scalable for large networks in terms of runtime for several reasons.

First, and most importantly, the number of database accesses is very large. Each expansion step loads the edges connected with the expanded vertex, typically resulting in one database query per step. Secondly, since vertices with the shortest distance to the query point are loaded first, space locality on disk is not guaranteed. Lastly, the full capacity of a disk block is not utilized in most cases. The edges will most likely not fill up the entire transferred block, and caching might not be very effective due to the lack of spatial locality during expansion. As a result both algorithms have a large I/O cost.

To decrease computation time, we suggest to load the data from the underlying spatial database in chunks into

the main memory. This will reduce the number of accesses to the data source and fills up transferred blocks better. Furthermore, if a clustered index is created for the network within the database and if the chunks are chosen wisely, I/O costs can be reduced since spatial locality on disk is regarded. We propose to use tiles to determine the chunks loaded from the database. They 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) [14]. 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 used throughout this paper, which also refers to the geographic data that lies inside a tile.

Definition 3.1 (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. Zoom level z=0 refers to the maximal extent of the network (for geographic data this usually 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 3.2 (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

Every vertex is mapped to exactly one tile region within a tile region matrix. Therefore, loading the edges for the next expansion step is straight forward.

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 OSM community [17]. 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 intersecting with this envelope by using the PostGIS operator "&&". Finally, all the edges connected to at least one of the vertices identified in the previous step are loaded.

Since tile regions are used, the proposed algorithm is called "Multimodal Incremental Network Expansion using Tile regions" (MINET). This naming schema coincides with the one of the algorithms MINE and MINEX from the papers [8] and [9]. The presented approach also allows vertex expiration, as it has been defined in [9], in order to free memory as early as possible. When using vertex expiration the algorithm is named "Multimodal Incremental Network Expansion using Tile regions and vertex eXpiration" (MINETX), which loads data in tiles and expires single vertices.

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 1 lists MINETX in pseudo-code. Just like MINEX does, it implements an incremental network expansions strategy.

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

```
   input  : q, d_max, s, t, z, N
   output : E^ra, V^ra
 1 C ← ∅;
 2 O ← {(v, 0, cnt_v)};
 3 while O ≠ ∅ ∧ (v, d_v, cnt_v) ← dequeue(O) ∧ d_v ≤ d_max do
 4     O ← O \ {v};
 5     C ← C ∪ {v};
 6     if indegree(v) = 0 then
           // v has not been loaded...
           // ...load all edges within the tile region defined by v and z
 7         T ← loadTileRegion(v, z) ;
 8         foreach (u', v') ∈ T do
 9             if u' ∉ O ∪ C then
10                 E_MM ← ∪{(u', ∞, cnt_u')};
11             if v' ∉ O ∪ C then
12                 E_MM ← ∪{(v', ∞, cnt_v')};
       // continue expansion just like MINEX would
13     foreach e = (u, v) ∈ E do
14         if u ∉ O ∪ C then
15             O ← O ∪ {(u, ∞, cnt_u)};
16         if u ∉ C then
17             d'_u ← τ(e, t − d_v) + d_v;
18             d_u ← min(d_u, d'_u);
19         if μ(θ(e)) ∈ {"csct", "csdt"} then
20             d'_u ← τ(e, t − d_v) + d_v;
21         cnt_u ← cnt_u − 1 ;
22         if u ∈ C ∧ cnt_u = 0 then
               // expire vertex
23             C ← C \ {u};
24         if μ(θ(e)) ∈ {"csct", "csdt"} then
25             if d'_u ≤ d_max then
26                 E^ra ← E^ra ∪ {(e, 0, λ(e))}
27             else
28                 E^ra ← E^ra ∪ {(e, o, λ(e))}, where d((e, o), q, t) = d_max
29     if cnt_v = 0 then
30         C ← C \ {v}
31     V^ra ← V^ra ∪ {(v, d_v)};
```

For Algorithm 1 it is assumed that the query point q coincides with a network vertex. If this is not true then q can easily projected to the nearest vertex within the network. The maximum travel duration within the resulting isochrone is determined by d_{max}, whereas the travel speed is defined by s. The arrival date and time at q is set by t. The zoom

level z needs to be passed as an input to the algorithm as well as the network N itself.

Just like MINEX does, the algorithm maintains two sets of vertices: closed vertices (C) that have already been expanded and open vertices (O) that have been encountered but are not yet expanded. For each vertex $v \in O \cup C$, the network distance to q, $d(v, q, t)$, and a counter, cnt_v, which keeps track of the number of outgoing edges that have not yet been traversed, is recorded. For an edge $e = (u, v)$, the function $\tau(e, t)$ computes the time-dependent transfer time required to traverse e. C starts as an empty set. O is initialized to v with $d_n(v, q, 0)$ and the number of connected edges.

During the expansion phase, vertex v with the smallest network distance is dequeued from O and added to C. If the tile region to which v belongs has not already been loaded, this is done now. As a result all edges that intersect with the tile region get loaded into main memory. All incoming edges, $e = (u, v)$, are retrieved from this memory and considered in turn. If vertex u is visited for the first time, it is added to O with a distance of ∞ and the number of outgoing edges, cnt_u. Then, the distance d'_u of u when traversing e is computed and the distance d_u is updated. If e is continuous ("*csct*" or "*csdt*"), the reachable part of e is added to the result. Edges of discrete type ("*dsct*" or "*dsdt*") produce no direct output, since only the vertices are accessible, and they are added when their "*csct*" edges are processed. Finally, cnt_u is decremented by 1; if u is closed and $cnt_u = 0$, u is expired and removed from C. Once all incoming edges of v are processed, the expiration and removal of v is checked. The algorithm terminates when O is empty or the network distance of the closest vertex in O exceeds d_{max}.

4 EVALUATION

In this section the results of an empirical evaluation of the proposed algorithm are described. The determination of the zoom level used by MINET and MINETX as well as the analyses of runtime and memory consumption are given.

4.1 Datasets

For the evaluation three different real-world datasets are used that vary in size, network-topology and the number of transportation vertices. They all were created using osmPti2mmds that is described in detail in [12].

Table 1 summarizes the datasets. The column "Size" denotes the network size, as it is stored in the database (in Mebibyte), whereas the other columns list the number of tuples stored. $|V|$ represents the total number of vertices, $|E|$ the total number of edges, $|E_{csct'}|$ the number of continuous (pedestrian) edges, $|E_{dsdt'}|$ of discrete (transportation) edges and $|S|$ the number of schedules. The average degree of the three networks is between 2.6 for BZ and 3.0 for WDC.

Dataset	Size	$\|V\|$	$\|E\|$	$\|E_{csct'}\|$	$\|E_{dsdt'}\|$	$\|S\|$
BER	571.8 MiB	201.3 k	557.99 k	538.19 k	13.81 k	2521.3 k
BZ	15.22 MiB	6.34 k	16.99 k	15.69 k	0.72 k	49.35 k
WDC	129.35 MiB	37.39 k	114.59 k	99.02 k	7.68 k	728.32 k

Table 1: Statistics about the datasets

The dataset BER corresponds to Berlin, the capitol of Germany. The data of the continuous network was retrieved in January 2017 from OSM and was enhanced with information about the public transportation system. The data is available from the "Verkehrsverbund Berlin-Brandenburg GmbH" (VBB). It is the biggest dataset in size and contains many different transportation modalities. Besides buses it also includes trams, trains and subways. The query point for this dataset is set to the "Brandenburg Gate" with an arrival time of 09.00 a.m. on a working day.

The dataset BZ represents the small city Bolzano in the north of Italy. Besides the pedestrian network that has been retrieved from OSM in January 2017 it also contains the public transportation system of the local transportation company "SASA (Società Autobus Servizi d'Area)", which is available as Open Data. It is given in a VDV452 format that was converted to GTFS regarding the official conversion instructions from SASA. The query point for the isochrone computation is a central square, namely the "Dominikanerplatz", with an arrival time of 09.00 a.m on a working day.

The dataset WDC describes the U.S. capitol Washington, D.C.. The pedestrian network again was retrieved in January 2017 from OSM. It was enriched by the schedules from the "Washington Metropolitan Area Transit Authority (WMATA)" which has been fetched from their developer portal. As query point the "Zero Milestone South of the White House" is used with an arrival time of 09.00 a.m. on a working day. In contrast to the dataset of Berlin this dataset reflects a grid based street network, while the other datasets do not (BER and BZ look more like a spider network).

4.2 System Description

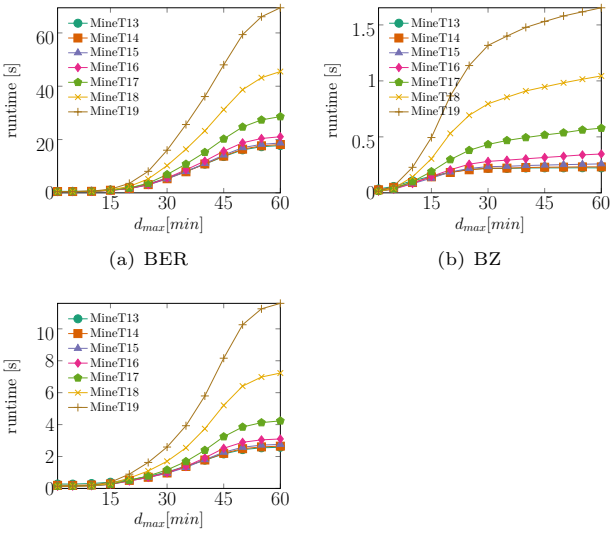
For creating the empirical results a system was used that is equipped with two Intel Xeon E5-2650 v2 processors running at 2.6 GHz (with a maximal turbo frequency of 3.2 GHz) and a total of 16 cores (8 cores per CPU). The main memory is DDR3-ECC RAM and 188 GB in size. The database used is PostgreSQL 9.5.5 with the PostGIS 2.2.4 extension enabled, while the operating system is CentOS 7.2.1511.

The tests listed in the following subsections were carried out 25 times. The plots represent the median of the results.

4.3 Determining the Zoom Level

The first examination during this evaluation is to determine the best zoom level for the fixed networks that later is used for runtime and memory evaluation. The runtimes of the zoom levels from 13 to 19 is plotted in Figure 1 to determine the optimal value of z. The memory consumption is not listed, since it is obvious that with growing zoom level it decreases while the number of database queries increases.

It can be seen that a large duration d_{max} implies the usage of smaller values for z. When aggregating all the runtimes over d_{max}, up to one hour the zoom levels z=14 and z=15 are the smallest in all three cases. In general, starting from a value of 16 they get slower as z increases. Decreasing the zoom level below 14 also increases runtime. The effect of



(a) BER (b) BZ

(c) WDC

Figure 1: Determination of the zoom level z.

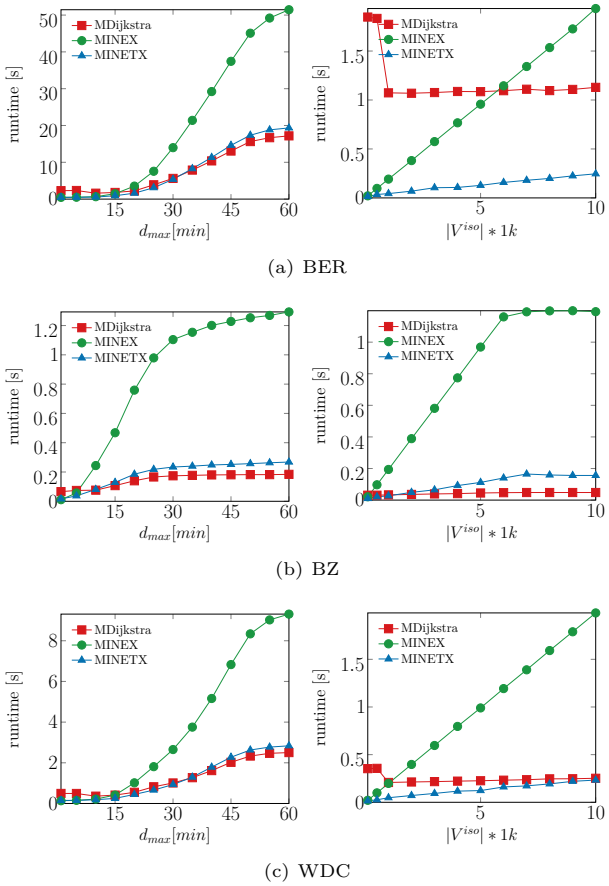
decreasing z is less dramatic, so it can not be easily seen from Figure 1. However, when looking at the raw numbers, the fastest zoom levels (after aggregation over d_{max}) are either 14 or 15, depending on the aggregation method (average or median) for all the datasets. Since a zoom level of z=15 saves a bit of memory and is comparably fast, z is set to 15 for the following evaluations.

4.4 Runtime

The results shown in Figure 2 are split into two parts. The left side of each subfigure shows the calculation with a limit of a specific isochrone duration d_{max} (given in minutes). The right side illustrates the calculation time needed to compute an isochrone of size $|V^{iso}|$ (given in number of vertices).

The first thing that can be noticed in the runtime results is that vertex expiration does not have a big influence on the evaluation system used. There is plenty of main memory available, so the runtimes for algorithms with and without vertex expiration are nearly the same (the difference of their runtimes is below one percent of time). This has been double-checked manually in the single test results and is true for all the runtimes. Therefore, only the results of the algorithms using vertex expiration are listed in Figure 2.

For the BER dataset, the runtime of the proposed algorithms is always less than the one for MINE and MINEX when using the isochrone's duration as limiting factor. It also stays below the runtime for MDijkstra until isochrones get bigger than about 30 minutes. An additional observation is that MINET and MINETX stay close to the runtime of MDijkstra even for larger isochrones. The same holds true for the WDC dataset, although the overall runtimes are less for this smaller dataset. For the smallest dataset of BZ the observation is still valid, with the exception that the algorithm MDijkstra becomes faster than the other algorithms very early (at an isochrone duration of about five minutes).



(a) BER

(b) BZ

(c) WDC

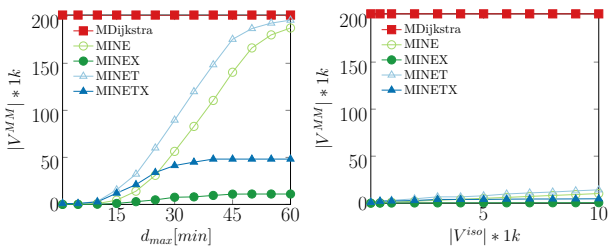
Figure 2: Comparison of calculation times.

When looking at the runtimes for limited isochrone size on the right side in the figure, the results are a bit different. The runtime of MDijkstra heavily depends on the dataset used. This is not true for neither MINE or MINEX, nor for MINET or MINETX. All these algorithms seem to grow linear with the size of the computed isochrone. However, the multiplication factor of the runtime of MINET (and MINETX) seems to be way better than the one of MINE (and MINEX).

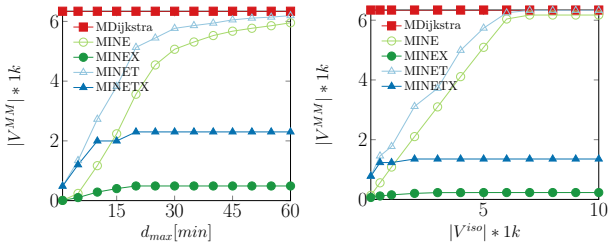
4.5 Memory Consumption

For the memory consumption it is again distinguished between evaluations that are bound by the maximal duration time of the isochrone (left side of Figure 3) and the maximal isochrone size (right side of Figure 3).

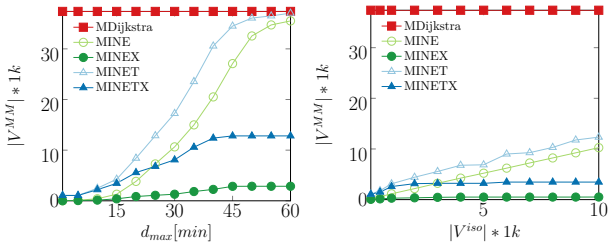
Figure 3 reveals some differences between the algorithms. It can be noticed that MDijkstra loads the whole network for all computations and therefore always needs the most memory. MINEX in difference is optimal in terms of memory consumption [9]. This results in the smallest memory footprint. MINETX does not perform as well as MINEX, but still significantly better than MDijkstra. It can be seen that data is loaded in chunks, since the memory consumption increases erratically at specific time points and stays constant



(a) BER



(b) BZ



(c) WDC

Figure 3: Memory consumption comparison.

over a specific time (until the next chunk needs to be loaded). This can best be observed in the dataset BZ where chunks that are relatively large when compared to the network size are loaded early.

Vertex expiration works well with both the MINEX and the MINETX algorithm. In contrast to the calculation times listed in Section 4.4 it has a big influence on the memory consumption. When compared to MINE and MINET, the expiration aware algorithms need less memory. This effect gets more and more important as the duration of the computed isochrone increases.

5 CONCLUSIONS

In this paper a new approach able to compute isochrones in multimodal spatial networks was presented. It loads data from the underlying data source using chunks, namely so called tile regions. This helps to reduce the number of calls to the database and also to minimize I/O costs. These regions also benefit from data locality, which also helps to better fill disk blocks during data transfer.

The empirical evaluation described in Section 4 shows that the proposed algorithm MINETX is always faster than the one presented by Gamper et al. in [9], but needs more main memory. In contrast to the algorithm MDijkstra developed by Bauer et al. in [3] it does not load the whole network into memory, nor keeps the computed isochrone in memory. Thus, its application areas are not limited by network size, neither by result size.

Further research regarding the presented method can be seen in varying the chunks loaded. The results when using tile regions look promising, but the quadratic nature of them is not optimal with respect to reachability. When loading a vertex at the very border of such a tile region, it is most likely the case that not all loaded vertices will later be expanded by the algorithm. Therefore, looking at circular ranges for loading chunks could be of interest for further investigations.

REFERENCES

[1] Aaron Antrim and Sean J Barbeau. 2013. The Many Uses of GTFS data. *Location-Aware Information Systems Laboratory* (2013).

[2] Crunchy Bagel. 2017. TransitFeeds. https://transitfeeds.com/. (2017).

[3] Veronika Bauer, Johann Gamper, Roberto Loperfido, Sylvia Profanter, Stefan Putzer, and Igor Timko. 2008. Computing Isochrones in Multi-Modal, Schedule-Based Transport Networks. In *Proc. of the 16th SIGSPATIAL conference*. ACM, 78:1–78:2.

[4] Moritz Baum, Valentin Buchhold, Julian Dibbelt, and Dorothea Wagner. 2016. Fast Exact Computation of Isochrones in Road Networks. In *Proc. of 15th SEA*. Springer, 17–32.

[5] Maria Antonia Brovelli, Marco Minghini, Monia Molinari, and Peter Mooney. 2016. Towards an Automated Comparison of OpenStreetMap with Authoritative Road Datasets. *Transactions in GIS* (2016).

[6] Edsger W Dijkstra. 1959. A Note on Two Problems in Connexion with Graphs. *Numer. Math.* (1959), 269–271.

[7] Heidelberg Institute for Geoinformation Technology (HeiGIT). 2017. OpenRouteService. http://openrouteservice.org/. (2017).

[8] Johann Gamper, Michael Böhlen, Willi Cometti, and Markus Innerebner. 2011. Defining Isochrones in Multimodal Spatial Networks. In *Proc. of the 20th CIKM conference*. ACM, 2381–2384.

[9] Johann Gamper, Michael Böhlen, and Markus Innerebner. 2012. Scalable Computation of Isochrones with Network Expiration. In *Proc. of 24th SSDBM conference*. ACM, Springer, 526–543.

[10] Jean-François Girres and Guillaume Touya. 2010. Quality Assessment of the French OpenStreetMap Dataset. *Transactions in GIS* (2010), 435–459.

[11] Ourania Kounadi. 2009. *Assessing the Quality of OpenStreetMap Data*. Master's thesis. University College of London.

[12] Nikolaus Krismer, Doris Silbernagl, Johann Gamper, and Günther Specht. 2016. osmPti2mmds–Erstellung von multimodalen Datensets aus OpenStreetMap und ÖPNV-Informationen. *AGIT* (2016).

[13] Mapzen. 2017. Transitland. https://transit.land/. (2017).

[14] Joan Masó, Keith Pomakis, and Núria Julià. 2010. *Web Map Tile Service Implementation Standard*. Technical Report. Open Geospatial Consortium.

[15] Peter Mooney, Padraig Corcoran, and Adam C Winstanley. 2010. Towards Quality Metrics for OpenStreetMap. In *Proc. of the 18th SIGSPATIAL conference*. ACM, 514–517.

[16] Pascal Neis and Dennis Zielstra. 2014. Recent Developments and Future Trends in Volunteered Geographic Information Research: The Case of OpenStreetMap. *Future Internet* (2014), 76–106.

[17] OpenStreetMap contributors. 2017. Slippy map tilenames. https://wiki.openstreetmap.org/wiki/Slippy_map_tilenames. (2017).

[18] OpenStreetMap Foundation. 2017. OpenStreetMap. https://www.openstreetmap.org/. (2017).

[19] Stefan Schröder, Peter Karich, and Michael Zilske. 2017. Graphhopper. https://graphhopper.com/. (2017).

[20] Stefan Wehrmeyer. 2017. Mapnificent. http://www.mapnificent.net/. (2017).

[21] Hermann Weiß and Judith Kammerer. 2017. NaturTrip. https://naturtrip.org/. (2017).