Scalable Computation of Isochrones with Network Expiration

Johann Gamper[1], Michael Böhlen[2], and Markus Innerebner[1]

[1] Free University of Bozen-Bolzano, Italy
[2] University of Zurich, Switzerland

Abstract. An isochrone in a spatial network is the possibly disconnected set of all locations from where a query point is reachable within a given time span and by a given arrival time. In this paper we propose an efficient and scalable evaluation algorithm, termed (MINEX), for the computation of isochrones in multimodal spatial networks with different transportation modes. The space complexity of MINEX is independent of the network size and its runtime is determined by the incremental loading of the relevant network portions. We show that MINEX is optimal in the sense that only those network portions are loaded that eventually will be part of the isochrone. To keep the memory requirements low, we eagerly expire the isochrone and only keep in memory the minimal set of expanded vertices that is necessary to avoid cyclic expansions. The concept of expired vertices reduces MINEX's memory requirements from $\mathcal{O}(|V^{iso}|)$ to $\mathcal{O}(\sqrt{|V^{iso}|})$ for grid and $\mathcal{O}(1)$ for spider networks, respectively. We show that an isochrone does not contain sufficient information to identify expired vertices, and propose an efficient solution that counts for each vertex the outgoing edges that have not yet been traversed. A detailed empirical study confirms the analytical results on synthetic data and shows that for real-world data the memory requirements are very small indeed, which makes the algorithm scalable for large networks and isochrones.

1 Introduction

Reachability analyzes are important in many applications of spatial network databases. For example, in urban planning it is important to assess how well a city is covered by various public services such as hospitals or schools. An effective way to do so is to compute isochrones. An *isochrone* is the possibly disconnected set of all locations from where a query point, q, is reachable within a given time span. When schedule-based networks, such as the public transport system, or time-dependent edge costs are considered, isochrones depend on the arrival time at q. Isochrones can also be used as a primitive operation to answer other spatial network queries that have to retrieve objects within the area of the isochrone. For instance, by joining an isochrone with an inhabitants database, the percentage of citizens living in the area of the isochrone can be determined without the need to compute the distance to individual objects.

Example 1. Figure 1 shows the 10 min isochrone at 09:15 pm for a query point (*) in Bozen-Bolzano. The isochrone consists of the bold street segments that cover all points

from where the query point is reachable in less than 10 minutes, starting at 09:05 pm or later and arriving at 09:15 pm or before. A large area around the query point is within 10 minutes walking distance. Smaller areas are around bus stops, from where the query point can be reached by a combination of walking and going by bus. The box in the lower left corner shows the number of inhabitants in the isochrone area.

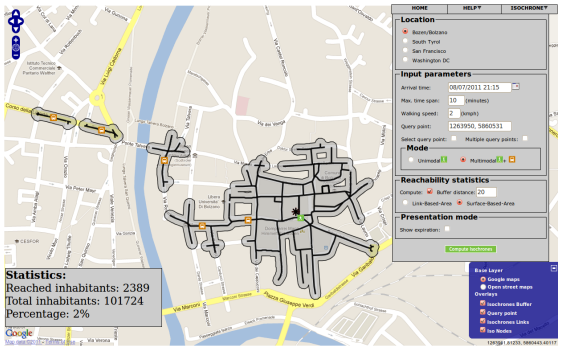


Fig. 1. Screenshot of an Isochrone.

We focus on isochrones in multimodal networks. Spatial networks 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, whereas in a *discrete space* network only the vertices can be accessed. *Continuous time* networks can be traversed at any point in time, *discrete time* networks follow an associated schedule. For instance, the pedestrian network is continuous in time and space, whereas public transport systems are discrete in both dimensions.

The paper proposes the Multimodal Incremental Network Expansion with vertex eXpiration (MINEX) algorithm. Starting from query point, q, the algorithm expands the network backwards along incoming edges in all directions until all space points that are within d_{max} from q are covered. Since only network portions are loaded that are part of the isochrone, the memory complexity of MINEX is independent of the network size. This yields a solution that scales to GIS platforms where a web server must be able to handle a large number of concurrent queries. The runtime is determined by the incremental loading of the network portions that are part of the isochrone. Our goal in terms of runtime is to be on a par with existing solutions for small to medium-size isochrones.

MINEX eagerly prunes the isochrone and keeps in memory only the expanded vertices that form the expansion frontier and are needed to avoid cyclic network expansions. These vertices must be updated as the expansion proceeds and the frontier moves outwards. Newly encountered vertices are added and vertices that will never be revisited, termed expired vertices, are removed. The removal of expired vertices reduces the memory requirements from $\mathcal{O}(|V^{iso}|)$ to $\mathcal{O}(\sqrt{|V^{iso}|})$ for grid and to $\mathcal{O}(1)$ for spider networks, respectively. Since the isochrone does not contain sufficient information to identify expired vertices, we propose an efficient strategy that counts for each vertex v

the outgoing edges that have not yet been traversed. When all these edges have been traversed, v will not be revisited and hence expires.

The technical contributions can be summarized as follows:

- We define *vertex expiration*, which allows to determine a minimal set of vertices that need to be kept in memory to avoid cyclic network expansions. Since an isochrone does not contain sufficient information to identify such vertices, we propose an efficient solution that counts the outgoing edges that are not yet traversed.
- We propose a scalable *disk-based multimodal network expansion algorithm*, MINEX, that is independent of the network size and depends only on the isochrone size. Its runtime is $\mathcal{O}(|V^{iso}|)$. The eager expiration of vertices reduces the memory requirements from $\mathcal{O}(|V^{iso}|)$ to $\mathcal{O}(\sqrt{|V^{iso}|})$ and $\mathcal{O}(1)$ for grid and spider networks, respectively.
- We show that MINEX 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.
- We report the results of an extensive empirical evaluation that shows the scalability of MINEX, confirms the analytical results for the memory requirements on synthetic data, and reveals an even more substantial reduction to a tiny and almost constant fraction of the network size on real-world data.

The rest of the paper is organized as follows. Section 2 discusses related work. In Section 3 we define isochrones in multimodal networks. Section 4 presents the MINEX algorithm. Section 5 reports the results of the empirical evaluation. Section 6 concludes the paper and points to future research directions.

2 Related Work

Isochrones have been introduced by Bauer et al. [3]. The algorithm suffers from a high initial loading cost and is limited by the available memory since the entire network is loaded in memory. Gamper et al. [8] provide a formal definition of isochrones in multimodal spatial networks, together with a disk-based multimodal incremental network expansion (MINE) algorithm that is independent of the actual network size but maintains the entire isochrone in memory. This paper proposes a new network expiration mechanism that maintains the minimal set of vertices that is required to avoid cyclic network expansions. The actual memory requirements turn out to be only a tiny fraction of the isochrone size. We conduct extensive experiments and compare our algorithm with other network expansion algorithms.

Marciuska and Gamper [18] start with an isochrone and present two different approaches to determine objects that are located within an isochrone. To capture objects that are not exactly on the streets but slightly outside (e.g., houses along streets), the basic idea is to first transform an isochrone represented as a subgraph into an isochrone represented as an area and to intersect this area with the objects relation.

Most network queries, including isochrones, are based on the computation of the shortest path (SP) among vertices and/or objects. Dijkstra's [6] incremental network expansion algorithm is the most basic solution and influenced many of the later works. Its major limitations are the expansion towards all directions and its main memory nature. The A^* algorithm [10] uses a lower bound estimate of the SP (e.g., Euclidean

distance) to get a more directed search with less vertex expansions. Other techniques have been proposed to improve the performance of SP and other network queries, including disk-based index structures and pre-processing techniques.

Papadias et al. [19] present two disk-based frameworks for computing different network queries: Incremental Euclidean Restriction (IER) repeatedly uses the Euclidean distance to prune the search space and reduce the number of objects for which the network distance is computed; Incremental Network Expansion (INE) is an adaptation of Dijkstra's SP algorithm. Deng et al. [5] improve over [19] by exploiting the incremental nature of the lower bound to limit the number of distance calculations to only vertices in the final result set. Almeida and Güting [4] present an index structure and an algorithm for kNN queries to allow a one-by-one retrieval of the objects on an edge.

Another strategy takes advantage of partitioning a network and pre-computing all or some of the SPs to save access and computation cost at query time. Examples are the partitioning into Voronoi regions and pre-computing distances within and across regions [16], shortest path quadtrees [21], and the representation of a network as a set of spanning trees with precomputed NN lists [11]. Other works divide a large network into smaller subgraphs that are hierarchically organized together with pre-computed results between boundary vertices [1, 13, 14].

For networks that are too large for exact solutions in reasonable time, efficient approximation techniques have been proposed, most prominently based on the landmark embedding technique and sketch-based frameworks [9, 17, 20, 22]. For a set of so-called landmark vertices the distance to all other vertices is pre-computed. At query time, the precomputed distances and the triangle inequality allow to estimate the SP.

The work in [7, 15] investigates time-varying edge costs, e.g., due to changing traffic conditions. There is far less work on schedule-based transportation networks and networks that support different transportation modalities [12]. Bast [2] describes why various speed up techniques for Dijkstra's SP algorithm are either not applicable or improve the efficiency only slightly in schedule-based networks.

The MINEX algorithm proposed in this paper leverages Dijkstra's incremental network expansion strategy for multiple transportation modes, and it applies eager network expiration to minimize the memory requirements. Most optimization techniques from previous work are not applicable to isochrones in multimodal networks, mainly due to the presence of schedule-based networks (cf. [2]) and the need to explore each individual edge, which makes search space pruning more difficult and less effective.

3 Isochrones in Multimodal Networks

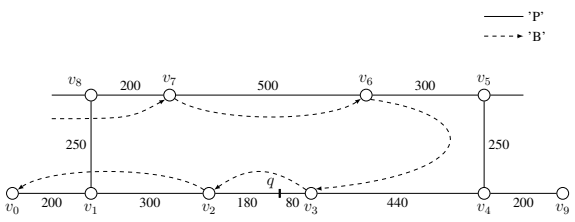
In this section we provide a formal definition of isochrones in multimodal spatial networks that support different transport modes.

Definition 1 (Multimodal Network). *A* multimodal network *is a seven-tuple* $\mathsf{N} = (G, R, S, \rho, \mu, \lambda, \tau)$. $G = (V, E)$ *is a directed multigraph with a set* V *of vertices and a multiset* E *of ordered pairs of vertices, termed edges.* R *is a set of transport systems.* $S = (R, TID, W, \tau_a, \tau_d)$ *is a schedule, where* TID *is a set of trip identifiers,* $W \subseteq V$*, and* $\tau_a : R \times TID \times W \mapsto \mathbb{T}$ *and* $\tau_d : R \times TID \times W \mapsto \mathbb{T}$

determine arrival and departure time, respectively ($\mathbb{T}$ is the time domain). Function $\mu : R \mapsto \{$'csct', 'csdt', 'dsct', 'dsdt'$\}$ assigns to each transport system a transport mode, and the functions $\rho : E \mapsto R$, $\lambda : E \mapsto \mathbb{R}^+$, and $\tau : E \times \mathbb{T} \mapsto \mathbb{R}^+$ assign to each edge transport system, length, and transfer time, respectively.

A multimodal network permits several transport systems, R, with different modalities in a single network: continuous space and time mode $\mu(.) =$ '*csct*', e.g., pedestrian network; discrete space and time mode $\mu(.) =$ '*dsdt*', e.g., the public transport system such as trains and buses; discrete space continuous time mode $\mu(.) =$ '*dsct*', e.g., moving walkways or stairs; continuous space discrete time mode $\mu(.) =$ '*csdt*', e.g., regions or streets that can be passed during specific time slots only. Vertices represent crossroads of the street network and/or stops of the public transport system. Edges represent street segments, transport routes, moving walkways, etc. The schedule stores for each discrete time ('*dsdt*', '*csdt*') transport system in R the arrival and departure time at the stop nodes for the individual trips. For an edge $e = (u, v)$, function $\tau(e, t)$ computes the time-dependent transfer time that is required to traverse e, when starting at u as late as possible yet arriving at v no later than time t. For discrete time edges, the transfer time is the difference between t and the latest possible departure time at u according to the given schedule in order to reach v before or at time t. This includes a waiting time should the arrival at v be before t. For continuous time edges, the transfer time is modeled as a time-dependent function that allows to consider, e.g., different traffic conditions during rush hours.

Example 2. Figure 2 shows a multimodal network with two transport systems, $R = \{$'P', 'B'$\}$, representing the pedestrian network with mode μ('P') = '*csct*' and bus line B with mode μ('B') = '*dsdt*', respectively. Solid lines are street segments of the pedestrian network, e.g., edge $e = (v_1, v_2)$ with $\rho(e) =$ 'P'. An undirected edge is a shorthand for a pair of directed edges in opposite directions. Pedestrian edges are annotated with the edge length, which is the same in both directions, e.g., $\lambda((v_1, v_2)) = \lambda((v_2, v_1)) = 300$. We assume a constant walking speed of 2 m/s, yielding a fixed transfer time, $\tau(e, t) = \frac{\lambda(e)}{2\,\text{m/s}}$. Dashed lines represent bus line B. An excerpt of the schedule is shown in Fig. 2(b), e.g., $TID = \{1, 2, \dots\}$, τ_a('B', 1, v_6) = τ_d('B', 1, v_6) = 05:33:00. The transfer time of a bus edge $e = (u, v)$ is computed as $\tau(e, t) = t - t'$, where $t' = \max\{\tau_d(\text{'B'}, tid, u) \mid \tau_a(\text{'B'}, tid, v) \leq t\}$ is the latest departure time at u.



(a) Network

R	TID	Stop	Arrival	Departure
B	1	v_7	05:31:30	05:32:00
B	1	v_6	05:33:00	05:33:00
	⋮	⋮	⋮	⋮
B	2	v_7	06:01:30	06:02:00
B	2	v_6	06:03:00	06:03:00
B	2	v_3	06:05:00	06:05:30

(b) Schedule

Fig. 2. Multimodal Network.

A *location* in N is any point on an edge $e = (u, v) \in E$ that is accessible. We represent it as $l = (e, o)$, where $0 \leq o \leq \lambda(e)$ is an offset that determines the relative position of l from u on edge e. A location represents 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 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 Fig. 2 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.

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. We generalize the length function for edge segments to $\lambda((e, o_1, o_2)) = o_2 - o_1$.

Definition 2 (Path, Path Cost). *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 $x_k = ((v_k, v_{k+1}), 0, o_d))$. With arrival time t at l_d, the* path cost *is*

$$\gamma(\langle x_1, \ldots, x_k \rangle, t) = \begin{cases} \tau(x_k, t) & k{=}1, \\ \gamma(\langle x_k \rangle, t) + \gamma(\langle x_1, \ldots, x_{k-1} \rangle, t - \gamma(\langle x_k \rangle, t)) & k{>}1. \end{cases}$$

The first and the last element in a path can be edge segments, whereas all other elements are entire edges. Since isochrones depend on the arrival time at the query point, we define the path cost recursively as the cost of traversing the last edge (segment), x_k, considering the arrival time t at the destination l_d, plus the cost of traversing $\langle x_1, \ldots, x_{k-1} \rangle$, where the arrival time at v_k (the target vertex of edge x_{k-1}) is determined as t minus the cost of traversing x_k. The cost of traversing a single edge is the transfer time τ. Edges along a path may belong to different transport systems, which enables the changing of transport system along a path.

Example 3. In Fig. 2, a path from v_7 to q is to take bus B to v_3 and then walk to q, i.e., $p(v_7, l_q) = \langle x_1, x_2, x_3 \rangle$, where $x_1 = (v_7, v_6)$ and $x_2 = (v_6, v_3)$ are complete edges and $x_3 = ((v_3, v_2), 0, 80)$ is an edge segment. With arrival time $t =$ 06:06:00, the path cost is $\gamma(p(v_7, l_q), t) =$ (06:03:00 − 06:02:00) + (06:05:20 − 06:03:00) + 80/2 = 240 s. To reach q at 06:06:00, the bus must arrive at v_3 no later than 06:05:20. Since the latest bus matching this constraint arrives at 06:05:00, we have a waiting time of 20 s at v_3.

The *network distance*, $d(l_s, l_d, t)$, from a source location l_s to a destination location l_d with arrival time t at l_d is defined as the minimum cost of any path from l_s to l_d with arrival time t at l_d if such a path exists, and ∞ otherwise.

Definition 3 (Isochrone). *Let $\mathsf{N} = (G, R, S, \mu, \rho, \lambda, \tau)$ with $G = (V, E)$ be a multimodal network, q be the query point with arrival time t, and $d_{max} > 0$ be a time span. An* isochrone*, $N^{iso} = (V^{iso}, E^{iso})$, is defined as the minimal and possibly disconnected subgraph of G that satisfies the following conditions:*

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

The first condition requires the vertices of the isochrone to be a subset of the network vertices. The second condition constrains an isochrone to cover exactly those locations that have a network distance to q that is smaller or equal than d_{max} Notice the use of edge segments in E^{iso} to represent edges that are only partially reachable. Whenever an edge e is entirely covered, we use e instead of $(e, 0, \lambda(e))$.

Example 4. In Fig. 3, the subgraph in bold represents the isochrone for $d_{max} = 5$ min and $t =$ 06:06:00. The numbers in parentheses are the network distance to q. Edges close to q are entirely reachable, whereas edges on the isochrone border are only partially reachable. For instance, (v_0, v_1) is only reachable from offset 80 to v_1. Bus edges are not included in the isochrone since intermediate points on bus edges are not accessible. Formally, the isochrone in Fig. 3 is represented as $N^{iso} = (V^{iso}, E^{iso})$ with $V^{iso} = \{v_0, \ldots, v_9\}$ and $E^{iso} = \{((v_0, v_1), 80, 200), ((v_8, v_1), 130, 250), (v_1, v_2), (v_2, v_1), (v_2, v_3), (v_3, v_2), (v_3, v_4), (v_4, v_3), ((v_5, v_4), 170, 250), ((v_9, v_4), 120, 200), ((v_5, v_6), 60, 300), ((v_7, v_6), 260, 500), ((v_6, v_7), 380, 500), ((v_8, v_7), 80, 200)\}$.

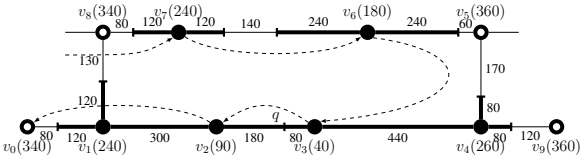


Fig. 3. Isochrone in Multimodal Network.

4 Incremental Network Expansion in Multimodal Networks

This section presents the multimodal incremental network expansion algorithm with vertex expiration (MINEX) for computing isochrones in multimodal networks.

4.1 Algorithm MINEX

Consider a multimodal network N, query point q with arrival time t_q, duration d_{max}, and walking speed s. The expansion starts from q and propagates backwards along the incoming edges in all directions. When a vertex v is expanded, all incoming edges $e = (u, v)$ are considered, and the distance of u to q when traversing e is incrementally computed as the distance of v plus the time to traverse e. The expansion terminates when all locations with a network distance to q that is smaller than d_{max} have been reached.

Algorithm 1 shows MINEX which implements this strategy. The multimodal network is stored in a database, and – as the expansion proceeds – the portions of the network that eventually will form the isochrone are incrementally retrieved. 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 $v \in O \cup C$, we record the network distance to q, d_v (abbrev. for $d(v, q, t_q)$), and a counter, cnt_v, which keeps track of the number of outgoing edges that have not yet been traversed. C is initialized to the empty set. O is initialized to v with $d_v = 0$ and the number of outgoing edges if q coincides with vertex v. Otherwise, $q = ((u, v), o)$ is an intermediate location, and O is initialized to u and v with the corresponding walking distance to q; the reachable segments of edges (u, v) and (v, u) are output.

Algorithm 1: MINEX($\mathsf{N}, q, t_q, d_{max}$)

```
1  C ← ∅;
2  if q coincides with v then O ← {(v, 0, cnt_v)};
3  else // q = ((u, v), o) = ((v, u), o')
4      O ← {(u, o/s, cnt_u), (v, o'/s, cnt_v)};
5      Output ((u, v), max(0, (o/s − d_max)s), o) and ((v, u), max(0, (o'/s − d_max)s), o');
6  while O ≠ ∅ and first element has distance ≤ d_max do
7      (v, d_v, cnt_v) ← first element from O;
8      O ← O \ {v};
9      C ← C ∪ {v};
10     foreach e = (u, v) ∈ E do
11         if u ∉ O ∪ C then O ← O ∪ {(u, ∞, cnt_u)};
12         d'_u ← τ(e, t_q − d_v) + d_v;
13         d_u ← min(d_u, d'_u);
14         cnt_u ← cnt_u − 1;
15         if u ∈ C ∧ cnt_u = 0 then C ← C \ {u};
16         if μ(ρ(e)) ∈ {'csct', 'csdt'} then
17             if d'_u ≤ d_max then Output (e, 0, λ(e));
18             else Output (e, o, λ(e)), where d((e, o), q, t_q) = d_max;
19     if cnt_v = 0 then C ← C \ {v};
20 return;
```

During the expansion phase, vertex v with the smallest network distance is dequeued from O and added to C. All incoming edges, $e = (u, v)$, are retrieved from the database 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 a '*csct*' or '*csdt*' edge, the reachable part of e is added to the result. A '*dsct*' and '*dsdt*' edges produce no direct output, since only the vertices are accessible, which are added when their incoming '*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 (more details on vertex expiration are below). 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}.

Example 5. Figure 4 illustrates a few steps of MINEX for $d_{max} = 5$ min, $t_q =$ 06:06:00, and $s = 2$ m/s. Bold lines indicate reachable network portions, solid black nodes are closed, and bold white nodes are open. The numbers in parentheses are the distance and the counter. Figure 4(a) shows the isochrone after the initialization step with $C = \{\}$ and $O = \{(v_2, 90, 3), (v_3, 40, 3)\}$. Vertex v_3 has the smallest distance

to q and is expanded next (Fig. 4(b)). The distance of the visited vertices is $d_{v_4} = 40 + 440/2 = 260\,\text{s}$ and $d'_{v_2} = 40 + 260/2 = 140\,\text{s}$, which does not improve the old value $d_{v_2} = 90\,\text{s}$. For the distance of v_6, we determine the required arrival time at v_3 as $t = t_q - d_{v_3} = 06{:}06{:}00 - 40\,\text{s} = 06{:}05{:}20$ and the latest bus departure at v_6 as 06:03:00, yielding $d_{v_6} = 40 + (06{:}05{:}20 - 06{:}03{:}00) = 180\,\text{s}$. After updating the counters, the new vertex sets are $C = \{(v_3, 40, 3)\}$ and $O = \{(v_2, 90, 2), (v_6, 180, 2), (v_4, 260, 2)\}$. Next, v_2 is expanded as shown in Fig. 4(c)). Figure 4(d) shows the isochrone after the termination of the algorithm; the gray vertex v_3 is expired.

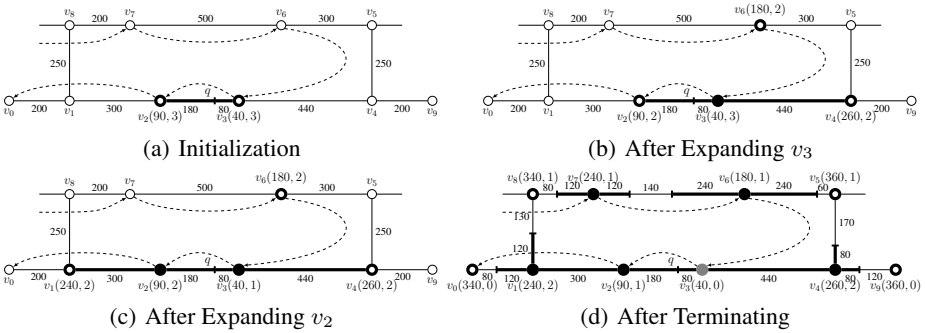


Fig. 4. Stepwise Computation of N^{iso} for $d_{max} = 5\,\text{min}$, $s = 2\,\text{m/s}$, and $t_q = 06{:}06{:}00$.

Notice that an algorithm that alternates between (completely) expanding the continuous network and (completely) expanding the discrete network is sub-optimal since many portions of the network would be expanded multiple times. We empirically evaluate such an approach in Sec. 5.

4.2 Expiration of Vertices

Closed vertices are needed to avoid cyclic network expansion. In order to limit the number of closed vertices that need to be kept in memory we introduce expired vertices (Def. 4). Expired vertices are never revisited in future expansion steps, hence they are not needed to prevent cyclic expansions and can be removed (Lemma 1). Isochrones contain insufficient information to handle vertex expiration (Lemma 2). Therefore, MINEX uses a counter-based solution to correctly identify expired vertices and eagerly expire nodes during the expansion (Lemma 3).

To facilitate the discussion we introduce a couple of auxiliary terms. For a vertex v, the term *in-neighbor* refers to a vertex u with an edge (u, v) and the term *out-neighbor* refers to a vertex w with an edge (v, w). Recall that the status of vertices transitions from open (O) when they are encountered first, to closed (C) when they are expanded, and finally to expired (X) when they are expired; the sets O, C, and X are pairwise disjoint.

Definition 4 (Expired Vertex). *A closed vertex, $u \in C$, is* expired *if all its out-neighbors are either closed or expired, i.e., $\forall v((u, v) \in E \Rightarrow v \in C \cup X)$.*

Example 6. Consider the isochrone in Fig. 4(d). Vertex v_3 is expired since v_2 and v_4 are closed, and v_3 has no other out-neighbors. In contrast, v_2 is not yet expired since the out-neighbor v_0 is not yet closed (and the expansion of v_0 leads back to v_2).

Lemma 1. *An expired vertex u will never be revisited during the computation of the isochrone and can be removed from C without affecting the correctness of MINEX.*

Proof. There is only one way to visit a vertex u during network expansion: u has an out-neighbor v (that is connected via an edge $(u, v) \in E$) and $v \in O$; the expansion of v visits u. Since according to Def. 4 all of u's out-neighbors are closed or expired, and closed and expired vertices are not expanded (line 11 in Alg. 1), u cannot be revisited.

The identification of expired vertices according to Def. 4 has two drawbacks: (1) it requires a database access to determine all out-neighbors since not all of them might already have been loaded, and (2) the set X of expired vertices must be kept in memory.

Lemma 2. *If the isochrone is used to determine the expiration of a closed vertex, $u \in C$, the database must be accessed to retrieve all of u's out-neighbors, and X needs to be stored in memory.*

Proof. According to Def. 4, for a closed vertex u to expire we have to check that all out-neighbors v are closed or expired. The expansion of u loaded all out-neighbors v that have also an inverse edge, $(v, u) \in E$. For out-neighbors v that are not connected by an inverse edge, $(v, u) \notin E$, we have no guarantee that they are loaded. Therefore, we need to access the database to get *all* adjacent vertices. Next, suppose that X is not maintained in memory and there exists an out-neighbor v of u without an inverse edge, i.e., $(v, u) \notin E$. If v is in memory, its status is known. Otherwise, either v already expired and has been removed, or it has not yet been visited. In the former case, u shall expire, but not in the latter case, since the expansion of v (re)visits u. However, with the removal of X we loose the information that these vertices already expired, and we cannot distinguish anymore between not yet visited and expired vertices.

Example 7. The isochrone does not contain sufficient information to determine the expiration of v_2 in Fig. 4(c). While v_1 and v_3 are loaded and their status is known, the out-neighbor v_0 is not yet loaded (and actually violates the condition for v_2 to expire). To ensure that *all* out-neighbors are closed, a database access is needed. Next, consider Fig. 4(d), where v_3 is expired, i.e., $X = \{v_3\}$. To determine the expiration of v_2, we need to ensure that $v_3 \in C \cup X$. If X is removed from memory, the information that v_3 is already expired is lost. Since v_3 will never be revisited v_2 will never expire.

To correctly identify and remove all expired vertices without the need to access the database and explicitly store X, MINEX maintains for each vertex, u, a counter, cnt_u, that keeps track of the number of outgoing edges of u that have not yet been traversed.

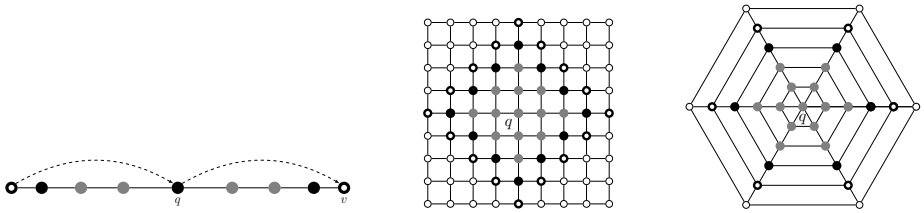
Lemma 3. *Let cnt_u be a counter associated with vertex $u \in V$. The counter is initialized to the number of outgoing edges, $cnt_u = |\{(u, v) \mid (u, v) \in E\}|$, when u is encountered for the first time. Whenever an out-neighbor v of u is expanded, cnt_u is decremented by 1. Vertex u is expired iff $u \in C$ and $cnt_u = 0$.*

Proof. Each vertex v expands at most once (when it is dequeued from O), and the expansion of v traverses all incoming edges (u, v) and decrements the counter cnt_u of vertex u by 1. Thus, each edge in the network is traversed at most once. When $cnt_u = 0$, vertex u must have been visited via all of its outgoing edges. From this we can conclude that all out-neighbors have been expanded and are closed, which satisfies the condition for vertex expiration in Def. 4.

Example 8. In the isochrone in Fig. 4(d), vertex v_3 is expired and can be removed since $cnt_{v_3} = 0$ and $v_3 \in C$. Vertex v_2 expires when v_0 is expanded and counter cnt_{v_2} is decremented to 0. Similar, vertex v_6 expires when v_5 is expanded.

Lemma 4. *Vertices cannot be expired according to an LRU strategy.*

Proof. We show a counter-example in Fig. 5(a), which illustrates a multimodal network expansion that started at q. Although q has been expanded and closed first, it cannot be expired because an edge from vertex v, which will be expanded later, leads back to q (and would lead to cyclic expansions). In contrast, the gray vertices that are expanded and closed after q can be expired safely.



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

Fig. 5. Network Expiration.

4.3 Properties

Vertex expiration ensures that the memory requirements are reduced to a tiny fraction of the isochrone. Figures 5(b) and 5(c) illustrate the isochrone size and MINEX's memory complexity for grid and spider networks, respectively. Solid black vertices (C) and vertices with a bold border (O) are stored in memory, whereas gray vertices are expired (X) and removed from memory. The following two lemmas provide a bound for the isochrone size and MINEX's memory complexity for these two types of networks. (Only the pedestrian mode is considered, though the results can easily be extended to multimodal networks.)

Lemma 5. *The size of an isochrone, $|V^{iso}|$, is $\mathcal{O}(d_{max}^2)$ for a grid network and $\mathcal{O}(d_{max})$ for a spider network and a central query point q.*

Proof. Consider the grid network in Fig. 5(b). Without loss of generality, we measure the size of an isochrone as the number of its vertices (i.e., open, closed, and expired vertices), and we assume a uniform distance of 1 between connected vertices. The size of an isochrone with distance $d = 1, 2, \ldots$ is given by the recursive formula $|V^{iso}|_d = |V^{iso}|_{d-1} + 4d$ with $|V^{iso}|_0 = 1$; $4d$ is the number of new vertices that are visited when transitioning from distance $d-1$ to d (i.e., the number of vertices at distance d that are visited when all vertices at distance d–1 are expanded). This forms an arithmetic series of second order (1, 5, 13, 25, 41, 61, ...) and can also be written as $|V^{iso}|_d = 1 + \sum_{i=0}^{d} 4i = 2d^2 + 2d + 1$, which yields $|V^{iso}| = \mathcal{O}(d_{max}^2)$.

Next, consider the spider network in Fig. 5(c). Without loss of generality, we assume a uniform distance of 1 between all adjacent vertices along the same outgoing line from q. It is straightforward to see that the size of the isochrone is $|V^{iso}| = deg(q) \cdot d_{max} + 1 = \mathcal{O}(d_{max})$, where $deg(q)$ is the degree of vertex q.

Lemma 6. *The memory complexity of MINEX is $|O \cup C| = \mathcal{O}(d_{max}) = \mathcal{O}(\sqrt{|V^{iso}|})$ for a grid network and $\mathcal{O}(1)$ for a spider network and a central query point q.*

Proof. Recall that MINEX keeps only the open and closed vertices, $O \cup C$, in memory. Consider the grid network in Fig. 5(b). By referring to the proof of Lemma 5, the cardinality of the open vertices at distance d can be determined as $|O|_d = 4d$ and the cardinality of the closed vertices as $|C|_d = 4(d-1)$. Thus, the memory requirements in terms of d_{max} are $|O \cup C| = \mathcal{O}(d_{max})$.

To determine the memory requirements depending on the size of the isochrone, $|V^{iso}|$, we use the formula for the size of an isochrone from the proof of Lemma 5 and solve the quadratic equation $2d^2 + 2d + 1 - |V^{iso}|_d = 0$, which has the following two solutions: $d_{1,2} = \frac{-2 \pm \sqrt{2^2 - 4 \cdot 2 \cdot (1 - |V^{iso}|_d)}}{2 \cdot 2} = \frac{-1 \pm \sqrt{2|V^{iso}|_d - 1}}{2}$. Since the result must be positive, $d = \frac{-1 + \sqrt{2|V^{iso}|_d - 1}}{2}$ is the only solution. By substituting d in the above formulas for open and closed vertices we get, respectively, $|O|_d = 4d = 4\frac{-1 + \sqrt{2|V^{iso}|_d - 1}}{2}$ and $|C|_d = 4(d-1) = 4(\frac{-1 + \sqrt{2|V^{iso}|_d - 1}}{2} - 1)$, which proves $|O \cup C| = \mathcal{O}(\sqrt{V^{iso}})$.

Next, we consider the spider network in Fig. 5(c) with the query point q in the centre. It is straightforward to see that the cardinality of the open and closed vertices is $|O \cup C| = 2 \cdot deg(q) = \mathcal{O}(1)$, where $deg(q)$ is the degree of vertex q.

Theorem 1. *Algorithm MINEX is optimal in the sense that all loaded vertices and 'csct'/'csdt' edges are part of the isochrone, and each of these edges is loaded and traversed only once.*

Proof. When a vertex v is expanded, all incoming edges $e = (u, v)$ are loaded and processed (Alg. 1, line 10). If e is a '*csct*'/'*csdt*' edge, the reachable portion of e (including the end vertices u and v) is added to the isochrone (line 16). While u might not be reachable, v is guaranteed to be reachable since $d_v \leq d_{max}$. In contrast, '*dsct*'/'*dsdt*' edges are not added since they are not part of the isochrone; only the end vertices u and v are accessible, which are added when the incoming '*csct*'/'*csdt*' edges are processed. Therefore, since each vertex is expanded at most once each edge is loaded at most once, and all loaded edges except '*dsct*'/'*dsdt*' edges are part of the isochrone.

5 Empirical Evaluation

5.1 Overview

Setup and Data Sets In the experiments we measure memory and runtime complexity of MINEX and compare it with the following alternative approaches: (1) Incremental network expansion INE [19], which incrementally loads the network but keeps all loaded vertices in memory. (2) Dijkstra's algorithm [6], which initially loads the entire network in memory. (3) Incremental Euclidean restriction IER [19], which instead of loading the entire network, uses the Euclidean lower bound property to incrementally load smaller network chunks as illustrated in Fig. 6. First, a chunk around q is loaded that contains all vertices that are reachable in walking mode (i.e., within distance $d_{max} * s$). After doing network expansion in memory, for all encountered bus stops a new (smaller) chunk is loaded, etc. Chunks are not re-loaded again if they are completely covered by network portions that are already in memory. (4) PGR, which is based on PostgreSQL's pgRouting and works similar to IER, but used the network distance instead of the Euclidean distance.

Fig. 6. Network Expansion in IER.

The multimodal networks and the schedules are stored in a PostgreSQL 8.4 database with the spatial extension PostGIS 1.5 and the PG routing extension. All experiments run in a virtual machine (64bits) on a Dual Processor Intel Xeon 2.67 GHz with 3 GB RAM. The algorithms were implemented in Java using the JDBC API to communicate with the database.

We test threes real-world networks that are summarized in Table 1 and described below. Size is the network size, and $|V|$, $|E|$, $|E_{'cscf'}|$, $|E_{'dsdt'}|$, and $|S|$ are the number of vertices, edges, pedestrian edges, edges of different means of transport, and schedule entries, respectively. The size is given in Megabyte, whereas the other columns show the number of tuples in K. We use also synthetic grid and spider networks like the one shown in Fig. 5(b) and 5(c). The walking speed in all experiments is set to 1 m/s. Isochrone size ($|V^{iso}|$) and the memory requirements ($|V^{MM}|$) are measured in terms of number of vertices. The other input parameters vary from dataset to dataset.

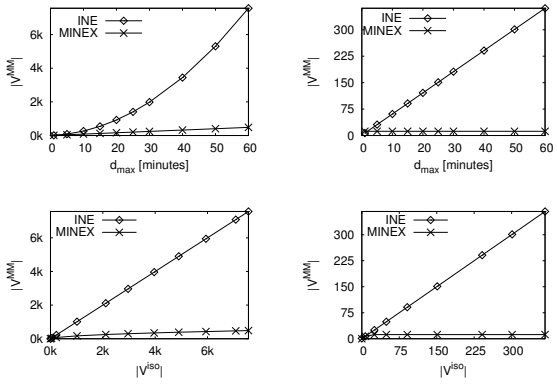
Table 1. Real-World Data Sets: Italy (IT), South Tyrol (ST), and San Francisco (SF).

Data	Size	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert E_{'cscf'}\rvert$	$\lvert E_{'dsdt'}\rvert$	$\lvert S\rvert$
IT	2,128	1,372.0	3,633.7	3,633.1	0.6	1.3
ST	137	77.7	197.8	182.4	9.4	179
SF	138	33.6	96.4	90.0	6.4	1,112

Summary of Experiments The first experiment in Sec. 5.2 measures memory and confirms that thanks to vertex expiration MINEX's memory requirements are only a small fraction of the isochrone size in all settings, whereas all other algorithms require significantly more. The second experiment in Sec. 5.3 shows that IER loads many edges multiple times, whereas MINEX loads each edge in the isochrone only once. The third experiment in Sec. 5.4 measures the runtime. For isochrones that are smaller than 9% of the network (which is frequently the case, especially for large and skewed regional networks) MINEX outperforms Dijkstra, whereas the latter is better for large isochrones, provided that the entire network fits into memory.

5.2 Memory Consumption

We begin with synthetic networks; only the pedestrian mode is used. The results confirm Lemma 5 and 6 and are shown in Fig. 7, where d_{max} and the isochrone size vary, respectively. For grid networks, MINEX's memory requirements grow linearly in d_{max} and with the square root in $|V^{iso}|$; INE's memory consumption corresponds to the isochrone size, i.e., $|V^{MM}| = |V^{iso}|$, and grows quadratically in d_{max}. In spider networks, the memory complexity is constant for MINEX and linear in d_{max} for INE.

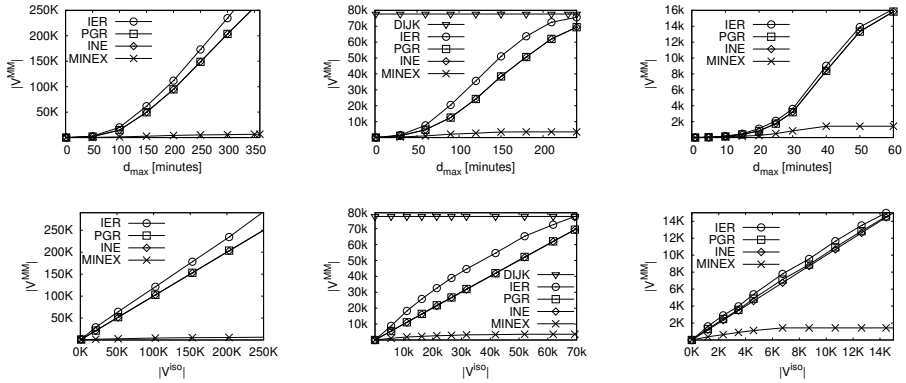


(a) Grid (b) Spider

Fig. 7. Memory Requirements in Synthetic Networks.

Figure 8 shows the memory complexity for the real-world data sets. We compare additionally with Dijkstra, IER, and PGR. As expected, MINEX's memory consumption is only a tiny fraction of the isochrone size, and it further decreases when the isochrone reaches the sparse network boundary. In contrast, the memory of INE, PGR and IER grows quadratically in d_{max} until the isochrone approaches the network border, where the growing slows down. For the IT data set this effect is not visible since d_{max} is too small. The memory of Dijkstra is equal to the size of the entire network, including vertices and edges, which can be inferred from Table 1, e.g., 2.1 GB for IT. To keep small values visible, the memory requirements of Dijkstra are not shown.

Figure 9 shows a 90 min isochrone for the ST network. Such regional or country-wide networks typically have an irregular network structure, varying density of vertices and edges, and fast wide-area transport systems. In this type of skewed networks, isochrones are characterized by many remote islands, and the size of isochrones is typically only a small fraction of the (comparably very large) network. Thus, Dijkstra is



(a) IT (b) ST (c) SF

Fig. 8. Memory Requirements in Real-World Data.

difficult to apply due to its high memory complexity, whereas MINEX requires only a tiny amount of memory (which in practice is even much smaller than the isochrone).

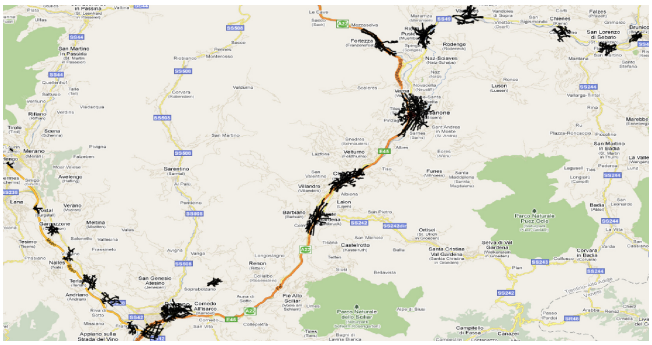
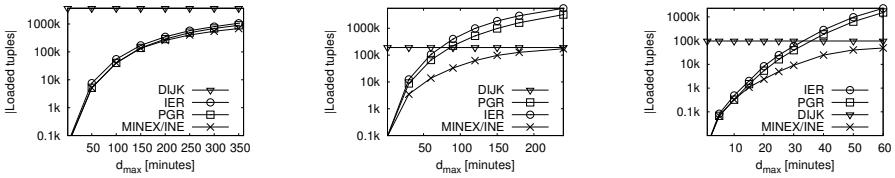


Fig. 9. Isochrone in the Regional ST Network.

5.3 Multiple Loading of Tuples

The experiment in Fig. 10 measures the total number of tuples (i.e., network edges) that are loaded from the database, using a logarithmic scale. MINEX and INE load the minimal number of tuples since each edge is loaded only once. They converge towards Dijkstra when the isochrone approaches the network size. In Fig. 10(a) IER and PGR load approximately the same number of tuples as MINEX since there is no overlapping in the areas that are retrieved by the range queries. This is because we have only the high-speed trains with very few stops. In contrast, in Fig. 10(b) and 10(c) the number of overlapping range queries is large, yielding many vertices and edges that are loaded multiple times. The reason here is the high density of public transport stops. Fig. 6 illustrates the overlapping of IER, which can be quite substantial.

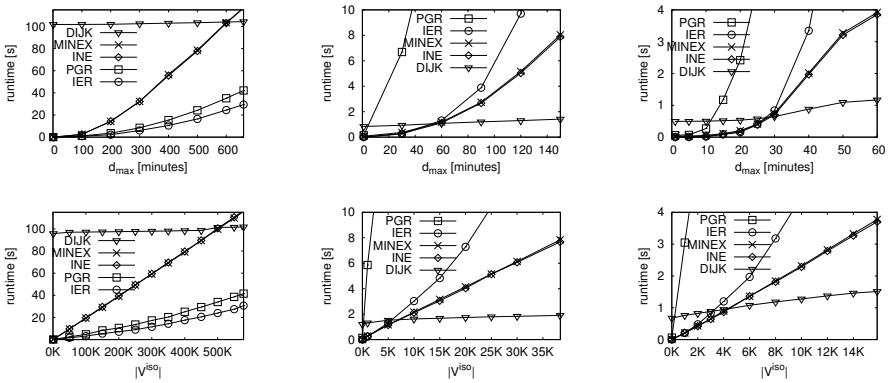


(a) IT (b) ST (c) SF

Fig. 10. Number of Loaded Tuples.

5.4 Runtime

Figure 11 shows the runtime depending on d_{max} and the isochrone size. For small values of d_{max} and small isochrones, Dijkstra has the worst performance due to the initial loading of the entire network. For large d_{max} and isochrones, Dijkstra (though limited by the available memory) is more efficient since the initial loading of the network using a full table scan is faster than the incremental loading in MINEX and INE, which requires $|V^{iso}|$ point queries.



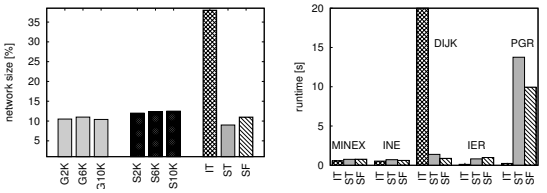
(a) IT (b) ST (c) SF

Fig. 11. Runtime in Real World Data.

The first experiment in Fig. 11(a) on the IT data set runs on a large skewed network with few and distantly located train stations. IER and PGR outperform Dijkstra since there are almost no overlappings due to the sparse number of train stations. INE and MINEX are slower than IER and PGR because of the larger number of database accesses (one access for each vertex expansion), but they are more efficient than Dijkstra. The break-even point occurs after 600 minutes when the size of the isochrone correspond to 38% of the network. Figure 11(b) shows the runtime of the regional network ST. Dijkstra outperforms INE and MINEX after a d_{max} of 60 minutes when the isochrone size corresponds to 9% of the network. IER and PGR collapse because of the large number of overlapping loaded areas in range queries. Figure 11(c) shows the run-

time in an urban network with a duration of one hour. The break-even point is reached at $d_{max} = 30$ min, which corresponds to 11% of the entire network.

In Fig. 12(a) we further analyze the break-even point for the real-world networks and for grid and spider networks with 2k, 6k, and 10k vertices (G2K, S2K, ...). We empirically determined that the break-even point is reached when the size of the isochrone is between 9–38% of the network size. Since the runtime of both Dijkstra and MINEX is dominated by the database accesses and the loading of the (sub)network, the break-even point occurs when the time for MINEX's incremental loading and the time for Dijkstra's initial loading cross. In terms of d_{max}, the break-even point varies depending on the walking speed and the frequency of the public transportation.



(a) Break-even Point (b) Network Independence

Fig. 12. Runtime.

Figure 12(b) confirms that MINEX is independent of the network size. We compute an isochrone of a fixed size $|V^{iso}| = 3.000$ for the different real-world data sets. The runtime of MINEX and INE is almost the same for all data sets. In contrast the runtime of Dijkstra depends directly from the network size. IER and PGR depends on the density of the public transport network.

6 Conclusion and Future Work

In this paper we introduced isochrones for multimodal spatial networks that can be discrete or continuous in, respectively, space and time. We proposed the MINEX algorithm, which is independent of the actual network size and depends only on the isochrone size. MINEX is optimal in the sense that only those network portions are loaded that eventually will be part of the isochrone. The concept of expired vertices reduces MINEX's memory requirements to keep in memory only the minimal set of expanded vertices that is necessary to avoid cyclic expansions. To identify expired vertices, we proposed an efficient solution based on counting the number of outgoing edges that have not yet been traversed. A detailed empirical study confirmed the analytical results and showed that the memory requirements are very small indeed, which makes the algorithm scalable for large networks and isochrones.

Future work points in different directions. First, we will investigate multimodal networks that include additional transport systems such as the use of the car. Second, we will investigate the use of various optimization techniques in MINEX as well as new approximation algorithms. Third, we will study the use of isochrones in new application scenarios.

References

1. V. Balasubramanian, D. V. Kalashnikov, S. Mehrotra, and N. Venkatasubramanian. Efficient and scalable multi-geography route planning. In *EDBT*, pages 394–405, 2010.
2. H. Bast. Car or public transport – two worlds. In *Efficient Algorithms*, volume 5760 of *LNCS*, pages 355–367, 2009.
3. V. Bauer, J. Gamper, R. Loperfido, S. Profanter, S. Putzer, and I. Timko. Computing isochrones in multi-modal, schedule-based transport networks. In *ACMGIS*, pages 1–2, 2008.
4. V. T. de Almeida and R. H. Güting. Using Dijkstra's algorithm to incrementally find the k-nearest neighbors in spatial network databases. In *SAC*, pages 58–62, 2006.
5. K. Deng, X. Zhou, H. T. Shen, S. W. Sadiq, and X. Li. Instance optimal query processing in spatial networks. *VLDB J.*, 18(3):675–693, 2009.
6. E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.
7. B. Ding, J. X. Yu, and L. Qin. Finding time-dependent shortest paths over large graphs. In *EDBT*, pages 205–216, 2008.
8. J. Gamper, M. H. Böhlen, W. Cometti, and M. Innerebner. Defining isochrones in multimodal spatial networks. In *CIKM*, pages 2381–2384, 2011.
9. A. Gubichev, S. Bedathur, S. Seufert, and G. Weikum. Fast and accurate estimation of shortest paths in large graphs. In *CIKM*, pages 499–508, New York, NY, USA, 2010. ACM.
10. P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. on Systems Science and Cybernetics*, SSC-4(2):100–107, 1968.
11. H. Hu, D. L. Lee, and J. Xu. Fast nearest neighbor search on road networks. In *EDBT*, pages 186–203, 2006.
12. R. Huang. A schedule-based pathfinding algorithm for transit networks using pattern first search. *GeoInformatica*, 11(2):269–285, 2007.
13. N. Jing, Y.-W. Huang, and E. A. Rundensteiner. Hierarchical encoded path views for path query processing: An optimal model and its performance evaluation. *IEEE Trans. Knowl. Data Eng.*, 10(3):409–432, 1998.
14. S. Jung and S. Pramanik. An efficient path computation model for hierarchically structured topographical road maps. *IEEE Trans. Knowl. Data Eng.*, 14(5):1029–1046, 2002.
15. E. Kanoulas, Y. Du, T. Xia, and D. Zhang. Finding fastest paths on a road network with speed patterns. In *ICDE*, 2006.
16. M. R. Kolahdouzan and C. Shahabi. Voronoi-based k nearest neighbor search for spatial network databases. In *VLDB*, pages 840–851, 2004.
17. H.-P. Kriegel, P. Kröger, M. Renz, and T. Schmidt. Hierarchical graph embedding for efficient query processing in very large traffic networks. In *SSDBM*, pages 150–167, 2008.
18. S. Marciuska and J. Gamper. Determining objects within isochrones in spatial network databases. In *ADBIS*, pages 392–405, 2010.
19. D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query processing in spatial network databases. In *VLDB*, pages 802–813, 2003.
20. M. Potamias, F. Bonchi, C. Castillo, and A. Gionis. Fast shortest path distance estimation in large networks. In *CIKM*, pages 867–876, 2009.
21. H. Samet, J. Sankaranarayanan, and H. Alborzi. Scalable network distance browsing in spatial databases. In *SIGMOD Conference*, pages 43–54, 2008.
22. M. Thorup and U. Zwick. Approximate distance oracles. In *STOC*, pages 183–192, New York, NY, USA, 2001. ACM.