Reachability Queries in Public Transport Networks

Bezaye Tesfaye
University of Salzburg
Department of Computer Science
Jakob-Haringer-Str. 2
5020 Salzburg, Austria
bezaye.belayneh@stud.sbg.ac.at

Nikolaus Augsten
University of Salzburg
Department of Computer Science
Jakob-Haringer-Str. 2
5020 Salzburg, Austria
nikolaus.augsten@sbg.ac.at

ABSTRACT

Given a query point in a spatial network, the reachability query retrieves all points of interest that are reachable from the query point in a specific amount of time respecting network constraints. Reachability queries have a number of interesting applications, and for road networks efficient solutions have been proposed. Road networks are time-independent, i.e., the cost for traversing an edge is constant over time. Efficient algorithms for road networks heavily rely on precomputing shortest paths. In public transport networks, however, the edge costs depend on schedules, which renders most solutions for road networks inefficient. In particular, shortest paths between node pairs cannot be easily precomputed because they change over time.

The goal of this work is to develop efficient solutions for reachability queries in public transport networks. The core idea is to partition the network into cells and compute time upper and lower bounds to traverse a cell. At query time, the reachable region is expanded cell by cell (instead of expanding edge by edge). All points of interest that are reachable using upper bound expansion are part of the result; all points that are not reachable in a lower bound expansion can safely be discarded; all other nodes are candidates and must be verified. This paper presents the expansion algorithm and and discusses interesting research directions regarding good network partitions, effective bounds, and efficient candidate verification.

Keywords

Spatial Network, Reachability Query, Public Transportation, Isochrones, Spatial Index, Spatial Network Databases, Spatio-temporal Databases

1. INTRODUCTION

Analyzing the reachability of geographic locations has many interesting application, for example, allocating hospitals and schools in urban planning, positioning franchise stores in geomarketing, or exploring the surroundings in mobile applications. In spatial networks, the reachability of a location does not only depend on the Euclidean distance, but also the restrictions and the structure of the network must be taken into account. For example, cars can only move on roads and must obey traffic rules, buses have schedules and can only be boarded at bus stops. Networks that allow different transport modes (e.g., pedestrian, car, and public transport) are called *multi-modal networks*.

Queries over spatial networks are essential for spatial and spatio-temporal databases [14, 18, 27, 29]. The integration of spatial networks into databases enables queries that respect network constraints and provides support for new fields of application. Many commerical and open source systems provide extensions for spatial networks, for example, Oracle Spatial, PostGIS for PostgreSQL, and ESRI ArcGIS Network Analyst. At the technical level, spatial networks pose a challenge regarding data modeling, indexing, and query processing. Examples of spatial network queries are shortest path queries, spatial joins, range queries, and nearest neighbor queries [19, 21, 23, 24].

A *reachability query* retrieves all points of interest (POI) in the network that reach (resp. are reachable from) a given query point within a given time frame (the *cost budget*) at a specific point in time. For example, in the query "return all students that can reach their school at 8am within 10 minutes either on foot or via public transport", the student homes are the POIs, the query point is the school, the cost budget is 10 minutes, and the time point is 8am.

There are two basic approaches to solve reachability queries: (a) compute the subset of the network that reaches the query point and intersect with the POI set, (b) compute the shortest path between each POI and the query point, and retain those POIs that are close enough. The first approach is beneficial for network areas with many result points; unfortunately, the algorithm must also expand network areas that do not contain any result. The efficiency of the second approach depends on the overall number of POIs and may require a large number of shortest path computations even for small result sizes.

For road networks, a number of solutions have been proposed, for example, Range Network Expansion (RNE), Range Euclidean Restriction (RER) [24], and Incremental Network Restriction [9]. Unfortunately, these approaches are not applicable to public transport networks: they rely on the Euclidean distance as a lower bound, which is not useful in the presence of schedules.

The problem of reachability queries is closely related to

28th GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 24.05.2016 - 27.05.2016, Noerten-Hardenberg, Germany.

the computation of shortest path queries. The fastest algorithms for the shortest path problem heavily rely on precomputation. If the shortest path between all pairs of nodes is precomputed, the shortest path computation is reduced to a single lookup; this approach, however, is not feasible for large networks. Algorithms like Contraction Hierarchy (CH) [12], Customized Route Planning (CRP) [8], and Transit Node Routing (TNR) [4] store the distances between a selected set of nodes which allow very fast query times with feasible index sizes in continent size networks. Unfortunately, these approaches assume road networks, which have constant cost for traversing an edge. Thus, there is a single shortest path between each pair of nodes. In public transport networks, the shortest path between two nodes depends on the query time. The edge cost is given by a schedule and varies over time. The cost fluctuation may be substantial: between a few minutes (during the day) and several hours (over night).

As stated by Bast et al. [2], "journey planning on public transportation systems, although conceptually similar [to journey planning in road networks], is a significantly harder problem due to its inherent time-dependent and multi-criteria nature". We identify two main problems: (1) The *time dependent edge costs* causes distances and shortest paths to change depending on the query time. (2) The *number of incident edges* of a single vertex in the network (e.g., train station) is much larger than the one in road network, where street junctions have few incident edges [2]. The time dependence of edge costs and the larger neighborhood of vertices render most approaches that rely on preprocessing infeasible for public transportation networks.

The goal of this work is to find an efficient solution for reachability queries in public transport networks using precomputation. Time and space for the precomputation should scale to continent size networks; the resulting index should support incremental updates in response to local network changes (e.g., new bus lines or changing schedules).

Our solution is based on graph partitioning. We split the transportation network into *cells* and precompute upper and lower bounds to traverse each cell. The precomputed bounds are used to efficiently expand the reachable region cell by cell (instead of edge by edge). Points of interest that are within a region found by an upper bound expansion are reachable, while points that are outside the lower bound region are not reachable. All points of interest that are within the lower bound but outside the upper bound region are candidates and must be verified.

2. RELATED WORK

In spatial networks, *reachability queries* are closely related to the shortest path (SP) problem, which has received much attention from the research community in recent years. Most of the current SP algorithms are based on Dijkstra's algorithm [10] or the A* algorithm [17]. These algorithms do not require any precomputation. Dijkstra's algorithm follows an expansion technique that visits edges in all possible direction until the target is reached, which makes the algorithm too slow for large networks. A* uses a heuristic (e.g., Euclidean distance) on top of Dijkstra's approach to direct the expansion. A good heuristic prevents unnecessary vertex expansion. Different optimization techniques have been proposed for Dijkstra's algorithm [25, 20, 7, 30] and the A* algorithm [15, 16].

By using extensive precomputation, the online time for SP queries can be substantially reduced. Highway Hierarchy (HH) [28] decreases the query time of Dijkstra's algorithm by up to three orders of magnitude [28]. The idea of HH is used by the Contraction Hierarchy algorithm [12, 13], which organizes vertices in hierarchies and applies a contraction technique to reduce the graph size for query processing. SPs are precomputed by adding new edges in the graph, which are leveraged at query time. This preprocessing technique of CH is used by SHARC (Shortcuts + ArcFlags) [5]. SHARC is considered one of the fastest unidirectional algorithm, while CH only works for bidirectional queries.

Another algorithm for SP computation mainly based on graph partitioning is Precomputed Cluster Distance (PCD) [22]. It partitions the graph into disjoint clusters and precomputes the SPs between all pairs of clusters. Transit node routing (TNR) [3] is based on an intuitive observation in road networks: all trips to a far destination typically share a small number of paths that contain important road junctions called *access nodes*. Using this idea, TNR first finds the *access nodes* in the graph and precomputes shortest path distances from and to the *access nodes*.

Customized Route Planning (CPR) [8] is based on a graph separator technique and is used for real time queries on road networks. The preprocessing of CRP is metric-independent, which is an advantage over CH and algorithms that follow similar precomputation techniques: CH provides fast query time, but is sensitive to small changes in the edge cost. Another algorithm based on graph separator is GRASP (Graph separators, RAnge, Shortest Path) [11], which uses a multilevel graph partitioning technique.

Most of the above algorithms have been evaluated in road and public transport networks, and the result is discussed in [2]. The evaluation shows that there is a big performance gap between the two types of networks. This is due to the time-dependent edge cost of public transportation networks, which make the precomputation effort of many algorithms infeasible for large networks.

To model a public transportation network as a graph, two approaches have been widely used: the time-expanded and the time-dependent approach. In time-expanded models, each arrival and departure event in each station (e.g., bus stop or train stop) is represented by a vertex, and an edge is used to represent the link between two events. The cost to traverse each edge is the time difference between the source and target event. In time-dependent models, each station is represented by a single vertex and an edge is used between two stations if there is a connection between them. A cost function that depends on the departure time at the source vertex is associated with each edge. The disadvantage of the first approach is the large size of the resulting graph. But, since each event is represented by a vertex, most of the SP algorithms designed for road networks can be applied in the resulting graph of this approach. See [26, 1] for a detailed explanation of both modelling techniques.

Range Euclidean Restriction (RER) and Range Network Expansion (RNE) [24] use an expansion technique similar to Dijkstra's algorithm to answer reachability queries. RNE first determines the reachable area and then intersects this area of the road network with the set of POIs. RER restricts the number of relevant POIs to the points that are reachable using Euclidean distance, i.e., ignoring the network. For these candidate points, the shortest path is computed in order to remove false positives. Deng et al. [9] reduce the number of shortest path computations to candidates in RER: similar to the A* shortest path algorithm [17], an Euclidean lower bound is maintained for each node that is expanded during the shortest path computation; the node that is closest to one of the candidates is expanded first, and the expansion stops when the lower bound to all remaining candidates exceeds the cost budget.

Bauer et al. [6] consider multimodal networks and do not rely on precomputation. They expand the query point using Dijkstra [10] and improve memory usage by expiring network nodes that have already been processed. The result of the computation is a so-called isochrone, which is the reachable portion of the network at a given point in time. Since all edges in the isochrone must be expanded, this approach does not scale to large networks and isochrones.

3. PROBLEM DEFINITION

We model the spatial networks as a directed graph $G = (V, E)$ with vertices V and edges E. Vertices are embedded in the plane, i.e., they have coordinates. A cost function $c(e, t)$ assigns each edge $e \in E$ a positive cost: the time to traverse the edge $e = (x, y)$ at time t (starting time at vertex x). The distance $d_t(u, v)$ between two nodes $u, v \in V$ is the minimum cost required to travel from u to v at time t ($d_t(u, v) = \infty$ if v is not reachable from u). A point p is in G if it is located on an edge or a vertex of G. The cost to traverse a partial edge is a linear fraction of the total cost.

DEFINITION 1 (REACHABILITY QUERY). *Given a spatial network $G = (V, E)$, a cost budget Δt, a set of points of interest P in G, a query point q in G, and the query time t. A reachability query computes all points of interest, $R \subseteq P$, that are reachable from q at start time t with cost Δt:*

$$R = \{p \in P \mid d_t(q, p) \leq \Delta t\}$$

The reverse reachability query *computes all points of interest from which q is reachable at arrival time t with cost Δt.*

The main objective of this work is to develop an algorithm for reachability queries in public transport networks. Current solutions for reachability queries are either restricted to road networks (i.e., time-independent edge cost) or do not scale to large networks.

4. A PARTITION-BASED APPROACH TO REACHABILITY QUERIES

In this section we sketch a technique for reachability queries based on network partitions and local precomputations within each partition.

We focus on public transport networks, which pose the main challenge to precomputation techniques. In a public transportation network graph, nodes are stations, e.g., bus stops or train stations. The time-dependent cost function that is associated to each edge represents the connection between two stations and depends on schedules and waiting times.

To simplify the discussion, the following presentation of our algorithm assumes undirected graphs. Later in this section we discuss possible extensions to directed graphs, which are a realistic model for a public transportation network.

4.1 Precomputation

4.1.1 Partitioning

We partition the graph into n *cells*. A cell C is a connected subgraph that consists of nodes $V_C \subseteq V$ and edges $E_C \subseteq E$, where E_C is the set of all edges that connect a pair of nodes in V_C. The cells are disjoint (i.e., for any pair $C_i \neq C_j$: $V_{C,i} \cap V_{C,j} = \emptyset$) and cover the nodes of the graph (i.e., $\cup_{i=1}^{n} V_{C,i} = V$). Edges that connect nodes from different cells are *border edges*, the end points of border edges are *border nodes*. The set of border nodes of cell C_i is denoted with B_i.

4.1.2 Graph Extension and Computation of Bounds

We add new nodes and edges to the graph. (1) In each cell C_i we connect all pairs of border nodes (b, b'), $b, b' \in B_i$. (2) We add a *virtual node* ϵ_i to each cell C_i and connect the virtual node to all border nodes. The virtual nodes are not part of the original graph. The cost between a border node $b \in B_i$ and the virtual node ϵ_i is the cost between b and the node in cell C_i that is most distant from b.

We compute time *upper bounds* and *lower bounds* between specific pairs of nodes. The upper bound is the shortest path between two nodes at the time of the slowest connection, i.e., the maximum shortest path over all points in time. The lower bound is the shortest path at the time of the fastest connection. We precompute upper and lower bounds for the following edges:

- *border nodes:* edges between all pair of border nodes within each cell;
- *virtual node:* edge between each border node $b \in B_i$ of a cell C_i and the virtual node ϵ_i of the cell;
- all *border edges.*

The index size (i.e., number of precomputed bounds) is

$$\text{index size} = 2|B_{i,j}| + \sum_{i=1}^{n} |B_i|(B_i + 1),$$

where $B_{i,j}$ is the set off all border edges in G, and B_i is the set of border nodes of cell C_i. Note that $B_{i,j}$ is a (hopefully small) subset of the nodes E in the original graph G. The summation adds up all upper and lower bounds that are computed within each cell. The summands only depend on the number border nodes in each cell, and the sum linearly increases with the number of cells. Thus, we expect the precomputation to scale to very large graphs.

The index can be incrementally updated in response to schedule changes, edge insertion, and edge removal. Only the updated edges and the cells that contain the updated edges are affected. The upper and lower bounds of all other cells and border edges remain unaffected by the update.

4.2 Candidate Generation

The core idea of our reachability algorithm is to expand cell by cell rather than expanding edge by edge. The expansion with upper bounds (slowest connection) defines a region $R_u \subseteq V$ that can always be reached within the given cost budget, independent of the starting time. All POIs in the upper bound region are part of the result set. The lower bound expansion (fastest connection) defines the region $R_l \subseteq V$ that is reachable under the assumption that each edge is traversed using the fastest connection, independent of the traversal time. All POIs outside the lower bound region R_l can safely be rejected since they are not part of the query result. The POIs between upper and lower bound region, $R_l \setminus R_u$ ($R_u \subseteq R_l$), are *candidates* which we must to verify. Compared to an approach that computes SP for all POIs, in our approach SP computations are limited to candidates. As will be discussed below, in some cases it might not be necessary to compute SP between a candidate and the query point in order to verify the candidate.

Next we discuss the expansion with upper and lower bounds. For the expansion algorithm we only need to consider border nodes, border edges, and the virtual nodes, i.e., all edges for which we have precomputed bounds. The expansion area is increased by units of cells, not nodes or edges.

4.2.1 Upper Bound Expansion

Assume a query point q that is located in cell $C_q = (V_q, E_q)$ with border nodes $B_q \subseteq V_q$ and virtual node ϵ_q. The expansion must first deal with cell C_q and then proceeds to neighboring cells until the budget is exhausted.

Initialization. We first expand cell C_q. The result is a (possibly empty) set of border nodes B of neighboring cells, i.e., $B \nsubseteq V_q$; each border node $b_i \in B$ has a time budget Δt_i. The expansion proceeds as follows:

1. *Expand cell C_q:* If all nodes in C_q are reachable from q with the cost budget Δt, then the upper bound region is initialized with the nodes of C_q, $R_u = V_q$; otherwise $R_u = \emptyset$. C_q is reachable from q if $\Delta t \geq 2\,\mathrm{ub}(b_q, \epsilon_q)$ for any node $b_q \in B_q$, where $\mathrm{ub}(x, y)$ is the upper bound between nodes x, y. The intuition is that we reach b_q from any node in C_q (i.e., also from q) at cost at most $\mathrm{ub}(b_q, \epsilon_q)$; from b_q we reach any other node in C_q with the same cost upper bound.

2. *Expand to neighboring cells.* Compute B, the set of all nodes that are reachable from a border node of C_q via a border edge and have a non-negative cost budget. The cost budget of node $b_i \in B$ that is reachable via border node $b_q \in B_q$ is $\Delta t_i = \Delta t - \mathrm{ub}(b_q, \epsilon_q) - \mathrm{ub}(b_q, b_i)$.

Expansion Step. Given a set of border nodes B outside R_u and a time budget Δt_i for each $b_i \in B$, the expansion proceeds as follows. For each node $b_i \in B$:

1. *Expand current cell:* Let C_i be the cell that contains border node b_i. If all nodes in C_i are reachable with the remaining cost budget at b_i, the cell is added to the upper bound region, i.e., if $\Delta t_i \geq \mathrm{ub}(b_i, \epsilon_i)$ then $R_u = R_u \cup V_i$. Otherwise only add border node b_i: $R_u = R_u \cup \{b_i\}$.

2. *Expand to neighboring cells.* Follow the edges between b_i and all other border nodes in B_i. The budget at node $b_j \in B_i \setminus \{b_i\}$ is $\Delta t_j = \Delta t_i - \mathrm{ub}(b_i, b_j)$. For all border nodes $b_j \in B_i$: (a) if $t_j \geq 0$, $R_u = R_u \cup \{b_j\}$; (b) for all nodes d reachable from b_j via a border edge: compute the cost budget $t_d = t_j - \mathrm{ub}(b_j, d)$ of node d; if $t_d \geq 0$, $B = B \cup \{d\}$.

3. Remove b_i from B.

The expansion step is repeated until the set of border nodes to be expanded is empty, $B = \emptyset$.

4.2.2 Lower Bound Expansion

The expansion with lower bounds is similar to the expansion with upper bounds. The main differences are:

1. The expansion uses lower bounds instead of upper bounds to compute the budgets of the nodes in B.

2. A cell C_i is added to the lower bound region R_l if (a) C_i contains the query node q (and $\Delta t > 0$), or (b) C_i contains a node of $b_i \in B$ with budget $\Delta t_i > 0$.

We need to expand all cells that are reachable via lower bounds with a non-zero budget. Even if we know that some nodes in the cell are unreachable (i.e., the lower bound to the virtual node exceeds the budget), we must make sure that the subset of nodes that is reachable is in the candidate set.

4.3 Extending the Algorithm

In the remaining section, we discuss extensions of the basic reachability algorithm. In particular, we discuss the extension to directed graphs, postulate criteria for a good partitioning of the graph into cells, introduce the concept of time-dependent bounds, and point to opportunities for candidate verification.

4.3.1 Directed Graphs

Public transportation networks will typically be modeled as *directed* graphs. In a directed graph, the bounds between a pair of nodes may depend on the direction. We extend our precomputation algorithm to compute *two* upper and lower bounds for each relevant node pair (one for each direction). For example, we distinguish the upper bound from a border node b_i to the virtual node ϵ_i from the upper bound in the reverse direction. Overall, the size of the precomputation will increase by a factor of two. The expansion algorithm must be adapted to the directed bounds, e.g., when expanding cell C_q, the condition will be $\Delta t \geq \mathrm{ub}(\epsilon_q, b_q) + \mathrm{ub}(b_q, \epsilon_q)$ (instead of $\Delta t \geq 2\,\mathrm{ub}(b_q, \epsilon_q)$, cf. Section 4.2.1).

In an undirected graph, all nodes in a cell are pairwise reachable. This may, in general, not hold for directed graphs, i.e., the bounds between a pair of nodes may be infinite. While still correct, the expansion algorithm would be less effective for the affected cells. Since we model public transport networks, this situation is very unlikely. If for some node there is a path only in one direction, we could reach that station but never leave it (or vice versa).

4.3.2 Effective Partitioning Strategy

So far we assumed that the partitioning of the graph into cells is given. Our algorithm is correct for any partitioning, but obviously the partitioning strategy will have an impact on the efficiency of the approach.

We identify a set of properties that an effective partitioning should satisfy:

- the number of border nodes per cell should be small;
- the overall number of border edges should be small;
- the difference between the upper and lower bounds of a cell should be small;
- all nodes in a cell should be pairwise reachable;
- different cells should be of similar size (i.e., homogeneous in terms of upper bound traversal time).

For the cell size, there is a balance to strike: Large cells improve the expansion speed since the processing time of a cell is independent of its size. Small cells allow finer grained expansion and lead to smaller candidate sets.

We expect that an algorithm for computing the optimal partitioning will have infeasible runtime. Thus we will target heuristics or approximations of the optimal algorithm.

4.3.3 Time-Dependent Bounds

In Section 4.1, we define the upper (lower) bound as the shortest path between two nodes at the time of the slowest (fastest) connection over all points in time. These bounds may be very loose for public transportation schedules, where the frequency of the connections will typically vary depending on the time of the day. For example, during rush hours the buses travel with the highest frequency, while there are no buses over night. This leads to lower bounds of minutes and an upper bounds of hours. For reachability queries during the day, the upper bound is too loose, leading to a very small upper bound region. Likewise, the lower bound is too small for departures in the evening, leading to large candidate sets.

Our solution are time-dependent bounds. We split the schedule into intervals and compute a different bound for each interval. During the expansion, we check the earliest (t_1) and latest (t_2) arrival time at a specific border node b. The upper bound is the maximum upper bound in the interval $[t_1, t_2]$; the lower bound is the minimum lower bound in the interval $[t_1, t_2]$. The interval of the arrival time is computed while expanding with the upper and lower bounds. Let b be a border node with time budget Δt^l during the lower bound expansion and time budget Δt^u during the upper bound expansion. With the query time t (i.e., the start time at the query node q) the earliest arrival time at node b is $t_1 = t + \Delta t - \Delta t_i^l$; the latest arrival time $t_2 = t + \Delta t - \Delta t_i^u$.

We have different options to split the schedule into intervals: We either use intervals of fixed length (e.g., 6 hours) or intervals of variable length. Variable length intervals will lead to tighter bounds since we can choose the intervals such that the bounds best approximate the true schedule. For example, we can create intervals that cover the rush hours, the day schedule, and the night schedule. Individual intervals can be chosen for each pair of nodes. The intervals may even be different between upper and lower bound for a given pair of nodes. The limiting factor is the precomputation size, which increases with each additional interval.

Table 1 shows an example of time-dependent bounds between a border node b and the virtual node ϵ. The 24-hours schedule is split into 4 intervals. The table shows the minimum and maximum travel time from b to any node in b's cell during the different time intervals. For example, between 6:01 and 12:00 the upper and lower bounds are 10min and 5min, respectively, while the bounds are 240min resp. 30min during the night. When we arrive at node b in the time interval $[11{:}17, 13{:}03]$, the upper bound is 15min and the lower bound is 5min.

Table 1: Time-dependent upper and lower bounds.

	0:00-6:00	6:01-12:00	12:01-18:00	18:01-24:00
ub	240min	10min	15min	30min
lb	30min	5min	10min	25min

4.3.4 Efficient Candidate Verification

The straightforward approach to verify a candidate node is by computing the shortest path between the query point and the candidate. Since we have precomputed lower bounds, we do not need to resort to Dijkstra's algorithm, but can use the efficient A^* algorithm and do a directed search.

In some cases we can reject or accept candidates without computing the shortest path. For example, the border nodes of the upper bound region that are in partially reachable cells can be used. These nodes have a remaining time budget. All candidates that are reachable from the border node within the given time budget (considering the shortest path between the border node and the candidate) are part of the result. Similarly, the border nodes of the lower bound region that do not reach all nodes in their cell can be used to reject candidates in that cell.

5. CONCLUSION

We propose a technique to compute reachability queries in public transportation networks by partitioning the graph into cells and use a novel expansion technique based on upper and lower bounds. The expansion algorithm generates a set of reachable points, a set of unreachable points that can be rejected without verification, and a set of candidates that must be verified. To get tighter bounds (closer upper and lower bounds), time-dependent upper and lower bounds are considered. This reflects the frequency patterns in public transportation schedules, which differ depending on the time of the day and the date (e.g., night buses, weekend schedules, holiday schedules). Better bounds lead to smaller candidate sets.

Different from the Dijkstra-like expansion techniques used in related work, we use precomputation and expect to dramatically reduce the number of shortest path computations for reachability queries. This will lead to better online query performance. The precomputation scales linearly with the network size, and local changes (e.g., schedule changes, new transport routes) require only a local update of the precomputed index.

As future work, we plan to develop an effective partitioning algorithm that satisfies the properties outlined in this paper. We will evaluate the proposed technique on real public transportation networks and empirically compare the performance to relevant competitors from related work. Further, we plan to generalize our solution for public transport networks to multimodal networks, which also include road and pedestrian edges.

6. REFERENCES

[1] H. Bast. *Efficient Algorithms : Essays Dedicated to Kurt Mehlhorn on the Occasion of His 60th Birthday*, volume 5760 of *Lecture Notes in Computer Science*, chapter Car or Public Transport – Two Worlds, pages 355–367. Springer, Berlin, Germany, 2009.

[2] H. Bast, D. Delling, A. V. Goldberg, M. Müller-Hannemann, T. Pajor, P. Sanders, D. Wagner, and R. F. Werneck. Route planning in transportation networks. *CoRR*, abs/1504.05140, 2015.

[3] H. Bast, S. Funke, and D. Matijevic. TRANSIT – ultrafast shortest-path queries with linear-time preprocessing. In *In 9th DIMACS Implementation Challenge [1]*, 2006.

[4] H. Bast, S. Funke, P. Sanders, and D. Schultes. Fast Routing in Road Networks with Transit Nodes. *Science*, 316(5824):566, Apr. 2007.

[5] R. Bauer and D. Delling. Sharc: Fast and robust unidirectional routing. *J. Exp. Algorithmics*, 14:4:2.4–4:2.29, Jan. 2010.

[6] 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 ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '08, pages 78:1–78:2, New York, NY, USA, 2008. ACM.

[7] A. Crauser, K. Mehlhorn, U. Meyer, and P. Sanders. A parallelization of dijkstra's shortest path algorithm. In L. Brim, J. Gruska, and J. Zlatu?ka, editors, *Mathematical Foundations of Computer Science 1998*, volume 1450 of *Lecture Notes in Computer Science*, pages 722–731. Springer Berlin Heidelberg, 1998.

[8] D. Delling, A. V. Goldberg, T. Pajor, and R. F. Werneck. Customizable route planning. In *Experimental algorithms*, pages 376–387. Springer, 2011.

[9] 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.

[10] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, 1(1):269–271, Dec. 1959.

[11] A. Efentakis and D. Pfoser. Grasp. extending graph separators for the single-source shortest-path problem. In A. Schulz and D. Wagner, editors, *Algorithms - ESA 2014*, volume 8737 of *Lecture Notes in Computer Science*, pages 358–370. Springer Berlin Heidelberg, 2014.

[12] R. Geisberger, P. Sanders, D. Schultes, and D. Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *Proceedings of the 7th International Conference on Experimental Algorithms*, WEA'08, pages 319–333, Berlin, Heidelberg, 2008. Springer-Verlag.

[13] R. Geisberger, P. Sanders, D. Schultes, and C. Vetter. Exact routing in large road networks using contraction hierarchies. *Transportation Science*, 46(3):388–404, 2012.

[14] B. George and S. Shekhar. Spatial network databases. In *Encyclopedia of Database Systems*, pages 2714–2719. Springer, 2009.

[15] A. V. Goldberg and C. Harrelson. Computing the shortest path: A* search meets graph theory. Technical Report MSR-TR-2004-24, Microsoft Research, Vancouver, Canada, July 2004.

[16] A. V. Goldberg and C. Harrelson. Computing the shortest path: A search meets graph theory. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '05, pages 156–165, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics.

[17] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, July 1968.

[18] V. Kanjilal and M. Schneider. Spatial network modeling for databases. In *Proceedings of the 2011 ACM Symposium on Applied Computing (SAC), TaiChung, Taiwan, March 21 - 24, 2011*, pages 827–832, 2011.

[19] M. R. Kolahdouzan and C. Shahabi. Continuous k-nearest neighbor queries in spatial network databases. In *STDBM*, pages 33–40. Citeseer, 2004.

[20] U. Lauther. An extremely fast, exact algorithm for finding shortest paths in static networks with geographical background. In *Geoinformation und Mobilität – von der Forschung zur praktischen Anwendung*, volume 22, pages 219–230, 2004.

[21] F. Li, D. Cheng, M. Hadjieleftheriou, G. Kollios, and S.-H. Teng. On trip planning queries in spatial databases. In *Proceedings of the 9th International Conference on Advances in Spatial and Temporal Databases*, SSTD'05, pages 273–290, Berlin, Heidelberg, 2005. Springer-Verlag.

[22] J. Maue, P. Sanders, and D. Matijevic. Goal-directed shortest-path queries using precomputed cluster distances. *J. Exp. Algorithmics*, 14:2:3.2–2:3.27, Jan. 2010.

[23] S. Nutanong and H. Samet. Memory-efficient algorithms for spatial network queries. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, pages 649–660, April 2013.

[24] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query processing in spatial network databases. In *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29*, VLDB '03, pages 802–813. VLDB Endowment, 2003.

[25] I. S. Pohl. Bi-directional search. *Machine Intelligence*, 6:127–140, 1971.

[26] E. Pyrga, F. Schulz, D. Wagner, and C. Zaroliagis. Efficient models for timetable information in public transportation systems. *J. Exp. Algorithmics*, 12:2.4:1–2.4:39, June 2008.

[27] P. Rigaux, M. Scholl, and A. Voisard. *Spatial databases: with application to GIS*. Morgan Kaufmann, 2001.

[28] P. Sanders and D. Schultes. Engineering highway hierarchies. In *Proceedings of the 14th Conference on Annual European Symposium - Volume 14*, ESA'06, pages 804–816, London, UK, UK, 2006. Springer-Verlag.

[29] S. Shekhar and S. Chawla. *Spatial databases: a tour*, volume 2003. prentice hall Upper Saddle River, NJ, 2003.

[30] M. H. Xu, Y. Q. Liu, Q. L. Huang, Y. X. Zhang, and G. F. Luan. An improved dijkstra's shortest path algorithm for sparse network. *Applied Mathematics and Computation*, 185(1):247–254, 2007.