Route Skyline Queries: A Multi-Preference Path Planning Approach

Hans-Peter Kriegel [1], Matthias Renz [2], Matthias Schubert [3]

Institute for Informatics, Ludwig-Maximilians-Universität München
Oettingenstr. 67, Munich, Germany
[[1]kriegel,[2]renz,[3]schubert@dbs.ifi.lmu.de]

***Abstract*— In recent years, the research community introduced various methods for processing skyline queries in multidimensional databases. The skyline operator retrieves all objects being optimal w.r.t. an arbitrary linear weighting of the underlying criteria. The most prominent example query is to find a reasonable set of hotels which are cheap but close to the beach. In this paper, we propose an new approach for computing skylines on routes (paths) in a road network considering multiple preferences like *distance*, *driving time*, *the number of traffic lights*, *gas consumption*, etc. Since the consideration of different preferences usually involves different routes, a skyline-fashioned answer with relevant route candidates is highly useful. In our work, we employ graph embedding techniques to enable a best-first based graph exploration considering route preferences based on arbitrary road attributes. The core of our skyline query processor is a route iterator which iteratively computes the top routes according to (at least one) preference in an efficient way avoiding that route computations need to be issued from scratch in each iteration. Furthermore, we propose pruning techniques in order to reduce the search space. Our pruning strategies aim at pruning as many route candidates as possible during the graph exploration. Therefore, we are able to prune candidates which are only partially explored. Finally, we show that our approach is able to reduce the search space significantly and that the skyline can be computed in efficient time in our experimental evaluation.**

I. INTRODUCTION

In recent years, the skyline operator has emerged as an important operation when searching a database for a set of result objects being ranked by various user preferences. Instead of considering a fixed weighting for a set of optimization criteria, the skyline operator computes all data objects that might offer an optimal result w.r.t. an arbitrary weighting of these criteria. The canonical example for this task is a traveler who wants to select a hotel w.r.t. two or more criteria, e.g. cost and distance to the city center. Since the user might not be able to determine a fixed ratio between both characteristics, the skyline operator now determines the set of all hotels that might be the best choice considering all possible ratios and thus, all possible user preferences. Technically, the quality of the result is modeled as the weighted sum of the object values over all considered criteria and the user preference is determined by a particular weight for each criterion. Thus, the higher this weight, the higher is the importance of the corresponding attribute to the user. The definition of the skyline implicitly defines the so-called domination relation between data objects. Object a dominates object b if a is as least as good as b w.r.t. all attributes and there exists at least one attribute where a is better than b. Thus, there cannot exist any weighted sum for which b would be more optimal than a.

There exists a considerable amount of efficient algorithms calculating skyline queries in databases. Most of these approaches are based on describing data objects as feature vectors where each dimension represents a given optimization criterion, e.g. [2], [18], [26], [31]. More recent approaches, started to consider skyline queries over more general object descriptions as well. For example, in [28] skyline computation in databases of uncertain objects is examined and in [4] the authors propose a general skyline computation method for arbitrary data objects that can be compared with a distance metric. Another important type of optimization criterion is the spatial distance in a road network. Going back to the hotel example, it often does not make sense to describe the distance to the city center w.r.t. the direct Euclidean distance. Instead, a better model would be to consider the path a person would have to walk from the hotel to the city center. If one of the compared positions is not previously known or even moving in the road network, the solution has to handle a dynamic attribute which has to be derived from the road network via shortest path computation before building the skyline.

Another aspect when searching for optimal paths in a road network is the selection of the underlying optimization criteria. Thus, the optimal path will vary depending on whether a user wants to reach the airport by the fastest or the shortest connection. Furthermore, the user might want to minimize highway fees or altitude differences. Thus, selecting a suitable path with respect to a given user preference is already an optimization problem in itself. Thus, a skyline over the set of all possible routes w.r.t. a set of relevant criteria will significantly improve the usability of route planning systems. Let us note that several route planning systems already employ multiple criteria when proposing a route to the user. All of these systems, determine routes by a single criterion only or employ a determined weighting between the criteria. However, none computes a skyline over all optimal combinations.

In this paper, we propose new algorithms for determining a skyline of routes w.r.t. multiple optimization criteria. The problem is highly dynamic because the skyline depends on previously unknown start and destination points. Precomputing all possible routes between all possible start and destination points is not a viable solution because the enormous amount

of routes would quickly exceed reasonable space limitations. Thus, our algorithm has to dynamically explore paths and prune those paths which cannot be extended into elements of the skyline as fast as possible. Our solution models the road network as a multi-attribute graph (MAG) storing a vector of different optimization criteria for each edge. To efficiently compute route skylines, we base our algorithm on a lower-bound forward estimation for each optimization criterion. If this optimistic forward estimation is already dominated by another route to the destination, we can stop further extending the route. Our first algorithm, called BRSC (Basic Route Skyline Computation) is similar to established skyline processing on feature vectors and uses exclusively this pruning criterion. However, BRSC can only prune a route r if there is an already-known route s to the destination which dominates the forward estimation of route r. Thus, this simple algorithm often has a large computational overhead because every path which could be potentially extended into a skyline path has to be kept and processed. Therefore, we introduce a second local pruning criterion based on the observation that any sub route of a skyline route needs to be pareto optimal as well. Thus, we propose a second algorithm called ARSC (Advanced Route Skyline Computing) employing both pruning criteria. Furthermore, the data structures employed by ARSC support pruning candidate paths without costly reorganizations. To summarize, we present the following contributions:

- We introduce route skylines queries on multi-attribute road networks as a new important skyline problem having applications in route planning and navigation systems.
- We define forward estimation for route skylines as a global pruning criterion for paths and introduce a simple algorithms (BRSC) for route skyline computation.
- We propose a local pruning criterion based on pareto optimal subroutes and introduce a more efficient skyline algorithm (ARSC) which employs both pruning criteria.

The rest of the paper is organized as follows. Section II surveys related work for skyline computation. In section III, we present basic definitions and briefly survey the underlying techniques for shortest path computation. The new pruning criteria and the proposed route skyline algorithms are introduced in section IV. Section V compares both new algorithms and demonstrates that our second algorithms is capable to calculate route skylines even on large road networks having over 170,000 nodes and links. Finally, section VI concludes the paper with a summary and some directions for future work.

II. Related Work

The skyline operator was introduced in [2]. Additionally, the authors propose block-nested-loop processing and an extended divide and conquer approach to process results for their new method. Since then, skyline processing has attracted considerable attention in the database community. In the following, we will briefly survey several algorithms for skyline processing in databases [31], [18], [26], [5], [25], extensions of the topic [3], [33], [29], [23], [4], [13] and finally, we will discuss dedicated methods employing skyline operators in road networks [12], [7], [15].

[31] proposes two progressive methods to improve the original solutions. The first technique employs Bitmaps and is directed towards data sets being described with low cardinality domains. In other words, each optimization criterion is described by a small set of discrete attribute values. Other solutions for this scenario are proposed in [5] and [25]. Since the attribute values of a path are summed up over an arbitrary amount of edges, this scenario is not relevant to our problem. The second technique proposed in [31] is known as index method and divides the data set into d sorted lists for d optimization criteria. [18] introduces the nearest neighbor approach which is based on an R-Tree [11]. This approach starts with finding the nearest neighbor of the query point which has to be part of the skyline. Thus, objects being dominated by the nearest neighbor can be pruned. Afterwards, the algorithm recursively processes the remaining sections of the data space and proceeds in a similar way. A problem of this approach is that these remaining parts might overlap and thus, the result has to be kept consistent. To improve this approach, [26] proposes a branch and bound approach called (BBS) which is guaranteed to visit each page of the underlying R-Tree at most once.

There is a large variety on methods that extend the basic skyline operator to multiple application types and settings. Furthermore, there are several post processing methods grouping or selecting the resulting skyline points. In [3] k-dominant skylines are proposed which generalize the dominance relationship by requiring that a point needs to improve all other points in at least k attributes. [33] and [21] discuss cube operators which allow the efficient parallel computation of skylines for subsets of all known optimization criteria. In [4] the authors examine dynamic skyline computation in general metric spaces. There exist several techniques for post-processing the result of skyline queries for the case that the number of skyline points becomes too large to be manually explored. The method proposed in [34] deals with the discovery of strong skyline points. [23] demonstrates that the selection of the k most representative skyline objects is a non trivial task and proposes a dynamic programming algorithm for the 2D case. Additionally, a polynomial time algorithm is presented for higher dimensionalities. In [29] the authors propose to group skyline points w.r.t. the subspace for which they are part of the skyline and they propose the algorithm Skyey for efficient computation. An extension of the skyline operator to uncertain data objets is presented in [28]. [22] and [32] examine the construction of skylines in a streaming environment. The problem of computing skylines in an distributed environment of multiple mobile devices is examined in [13].

There has already been some work considering the application of the skyline operator in a setting including road networks. In [7] the authors introduce the problem of multi-source skyline processing in road networks. The general task in this paper is to calculate a skyline of landmarks in a road network

that are compared w.r.t. their network distance to several query objects or persons traveling in the network. The proposed approach shares some similarity to our method because it is calculating a skyline based on a road network. However, the shortest paths in this approach describe distances and thus, can be considered as feature values for a fixed set of objects. In contrast, our approach ranks paths between a single pair of a starting point and a destination point instead of ranking single locations. Furthermore, our approach considers an arbitrary number of network properties whereas the method proposed in [7] considers only a single edge attribute, i.e. length. A final difference is that our approach computes pareto optimal paths whereas the method in [7] calculates a skyline of locations. In [12] the authors propose in-route skyline processing in road networks. Assuming a user is moving along a predefined route to a known destination, the algorithm processes minimal detours to sets of landmarks being distributed along the path. For example, a user might want to visit a super market or a gas station during his way from home to work. The approach is different to our setting as it works with single attribute edge-weights and ranks detours as skyline routes. In comparison, the approach discussed in this paper employs multi-attribute edge weights and determines a skyline of alternative routes. In [15] the authors discuss continuous skyline queries in road networks. In this work, it is assumed that a user is moving on a road network and wants to process the skyline over a set of locations like restaurants or hotels. To compute this skyline the method can rely on attributes which are considered as fixed, e.g. price, quality, etc.. Additionally, the algorithm has to consider one dynamic attribute, i.e. the distance to the considered locations which is continuously changing due to the moving query point. Since this method again ranks spatial location instead of routes, it is also concerned with a different problem. To conclude, all of these related topics deal with a completely different setting and different goals.

Route planning systems for road networks are usually based on finding the shortest path between two objects, e.g. computed by Dijkstra's algorithm [8] and its variants [6]. The A^*-algorithm [20] applies heuristics to prune the search space and to direct the graph expansion. Another approach is followed materialization techniques like [1], [14], [16]. However, these methods suffer from increasing storage cost. In [17] the authors divide the graph into regions and gather information whether an edge is on a shortest path leading to a specific region. All these approaches provide only efficient path computation based on one single road attribute, but cannot be used for a multi-preference based route planning.

The Euclidean distance between graph nodes/objects can be used to lower bound the network distance between two graph nodes as proposed for the *incremental Euclidean restriction* (IER) method introduced in [27]. Here, the Euclidean distance is used as filter in order to restrict the search space for the identification of the k-nearest neighbors of a query object. This approach works well only if the object neighborhood based on the Euclidean distance well approximates the object neighborhood based on the network distance which is not true in real spatial networks. To resolve this problem, the incremental network expansion method (INE) based on Dijkstra was proposed. In [30], one of the graph embedding technique from [24] is applied in order to estimate the network distance between two nodes and extended dynamic embedding for moving objects is presented. A severe drawback of the approach is that the embedded space involves 40 to 256 dimensions. The work of this paper is based on the network graph embedding originally proposed independently by two research groups [10], [9] and [19]. While the work in [10], [9] only explores a lower bound, the authors in [19] also derive an upper bound for the network distance. In addition, the authors in [10], [9] focus only on speeding up the shortest path computations whereas in [19], the authors propose a multi-step query processing framework for supporting proximity queries in traffic networks.

III. Route Skylines in Network Environments

In this section, we will specify the problem of route skylines and discuss the requirements to possible solutions. Therefore, we will begin with defining multi-attribute network graphs for representing a road network:

Definition 1 (Multi-attribute network graph (MAG)): A multi-attribute network graph (MAG) is a directed network graph $\mathcal{G}(V, E, W)$ with V denoting a set of vertices, $E \subset V \times V$ denoting a set of edges and $W \subset \mathbb{R}_+^d$ denoting a set of d-dimensional positive weight vectors. Since G is directed $e = (v_s, v_t) \neq (v_t, v_s) = \hat{e}$. Furthermore, let $\omega : E \Rightarrow \mathbb{R}^d$ be a mapping, assigning a weight vector ω to each edge $e \in E$. The l-th attribute of edge e is denoted as $\omega(e)_l$.

In our application, a MAG represents a road network and thus, the nodes correspond to crossings, the edges correspond to road segments and the weight vectors describe the considered attributes of each road segment. The attributes of a segment might represent the length, the maximum speed, the time to pass the segment, the number of pedestrian crossings or the maximum ascent etc.. In the following, we will assume that all attributes are positive and a small weight is more beneficial than a large weight. Let us note that this is no general limitation of our approach because our algorithm can be modified to handle other types of attributes as well. However, this would make the description unnecessary complicated.

A path p and its cost $cost_l(p)$ w.r.t. the l-th attribute in this network is specified as follows:

Definition 2 (Path and Cost of a Path): A path p is a sequence of nodes $(v_1, .., v_k)$ where the following conditions hold:

$$\forall 1 \leq i < k : \exists e \in E : e = (v_i, v_{i+1})$$
$$\forall i \neq j : n_i \neq n_j$$

The cost of path $p = (n_1, .., n_k)$ w.r.t. the l-th attribute is defined as follows:

$$cost^l(p) = \sum_{i=0}^{k-1} \omega((v_i, v_{i+1}))_l$$

While condition (1) describes a path as a walk, i.e. a connected sequence of edges, condition (2) specifies that no cycles are allowed in a path, i.e. the path visits no node twice. Considering our application it makes sense that a route a driver wants to select does not contain any circles because the predominant goal of driving is reaching a certain destination. Thus, we will only consider paths in the following. The cost of a path is summed up over the costs of all contained edges. For example, if the l-th attribute describes length, then the cost of path p correspond to the total length of the path.

Our algorithm aims at finding all paths that are optimal w.r.t. a given user preference. Thus, we have to specify a preference function mirroring this intention:

Definition 3 (Preference Function): Given the d-dimensional space of edge weights $W \subseteq \mathbb{R}^d_+$, the preference function $Pref_\Pi(p)$ for path p is defined as follows:

$$Pref_\Pi(p) = \sum_{l=1}^{d} \pi_l \cdot cost^l(p)$$

under consideration of the preference vector $\Pi = (\pi_1, \ldots, \pi_d) \in \mathbb{R}^d$.

The preference function is the weighted sum over all considered edge attributes W and the preference vector Π describes the importance of each attribute to the given user query. For example, if a user is only interested in the total length of a path, Π consists of a zero vector with the exception of a one for the attribute representing the segment length. As mentioned above, we assume that each attribute represents some type of cost and thus, the lower $Pref_\Pi(p)$ is, the better is path p . Correspondingly, path p is an optimal selection under consideration of the preference Π if its cost is minimal, or following the common terminology, if $Pref_\Pi(p)$ is a shortest path w.r.t. Π.

Definition 4 (Preference Shortest Path): Given two nodes $v_s, v_t \in V$, then path $\check{p} = (v_s, \ldots, v_t)$ is called shortest path or minimal path for the user preference Π iff $\forall p = (v_s, \ldots, v_t) : Pref_\Pi(\check{p}) \leq Pref_\Pi(p)$

Let us note that there might be more than one path sufficing this condition.

A. Shortest Path Calculation

The most well-known approach for shortest path computation was proposed by Dijkstra. Though this algorithm is an optimal solution when assuming that no additional information about the shortest path is available, it usually has to consider large portions of the graph. To further speed up shortest path computation, it is necessary to employ an optimistic approximation of the remaining distance to the destination for each point in the network. Adding this approximation to the cost of the currently explored path, we may decide that it is not possible to reach the destination via an extension of this path any faster than the present shortest path. Thus, we can prune the path if there is an already known path to the destination which is already shorter than the currently examined path. This is the core idea behind A^*-Search which explores paths in the order of the smallest optimistic approximation. A^*-Search can significantly reduce the number of nodes that have to be traversed for shortest path computation. Considering the length of each road segment as its cost, a simple solution for calculating a lower bound approximation is to employ Euclidean distance. Since there is no shorter way between two points than the direct line, the Euclidean distance will always be lower than or equal to the distance which has to be traversed on the road network. Thus, for this particular cost function the Euclidean distance allows A^*-Search. Unfortunately, this natural lower bound is not extendable to other criteria which are not strongly correlated to the spatial distance. Thus, for applying A^*- Search on an arbitrary preference function combining various, general optimization criteria, it is necessary to employ a more general approach.

To solve this problem, we employ a special form of Lipschitz embedding of the traffic network using singleton reference sets which we call *reference nodes* according to [19] (in [10], [9], these reference nodes are called *landmarks*). The embedding transforms the nodes of a given multi-attribute network graph into $d \times k$-dimensional vectors, where d denotes the dimensionality of the edge weight vectors of our graph and k the number of reference nodes. In the following, we will only consider the embedding according to a single road attribute l to simplify the presentation. Therefore, $d^l_{net}(u, v)$ denotes $cost^l(sp_{u,v})$ where $sp_{u,v}$ describes the shortest path between the nodes u and v w.r.t. attribute l.

Let $\mathcal{G} = (V, E, W)$ be a MAG and $V' = \langle v_{r_1}, \ldots, v_{r_k} \rangle \subseteq V$ be a subsequence of $k \geq 1$ reference nodes. The embedding, or transformation, of the native space V into a k-dimensional vector space $\mathbb{R}^k$ according to a road attribute l is a mapping $F^{V',l} : V \to \mathbb{R}^k$, where $|V'| = k$ is the dimensionality of the vector space. A *reference node embedding* of $\mathcal{G}$ based on $V' \subset V$ defines the function $F^{V',l}$ as follows:

$$\forall v \in V : F^{V',l}(v) = (F_1^{V',l}(v), \ldots, F_k^{V',l}(v))^{\mathbf{T}},$$

where $F_i^{V',l}(v) = d^l_{net}(v, v_{r_i})$ for $1 \leq i \leq k$ and $d^l_{net}(v, v_{r_i})$ denotes the network distance between node v and node v_{r_i} according to the corresponding road attribute l.

An example demonstrating the embedding of the network graph according to a road attribute l using the reference nodes $V' = \langle v_8, v_7 \rangle$ is depicted in Figure 1. Note, that our example does not show a complete embedding, rather we just displayed the embedding of a subset of graph nodes (those nodes are displayed using a cross instead of a point). The left graph shows the network graph with edge weights corresponding to one road attribute. Nodes, $v_7 \in V'$ and $v_8 \in V'$ are selected as reference nodes. On the right side, the (partial) embedding according to V' is depicted.

When constructing the embedding for a network graph, we have to compute for each node and each attribute the shortest paths to all reference nodes. Since the graph structure remains fixed, the embedding is processed off-line and the results are stored with each node. To compute the graph embedding w.r.t reference node v_r, we perform a graph expansion based on Dijkstra starting from v_r until all nodes have been visited.

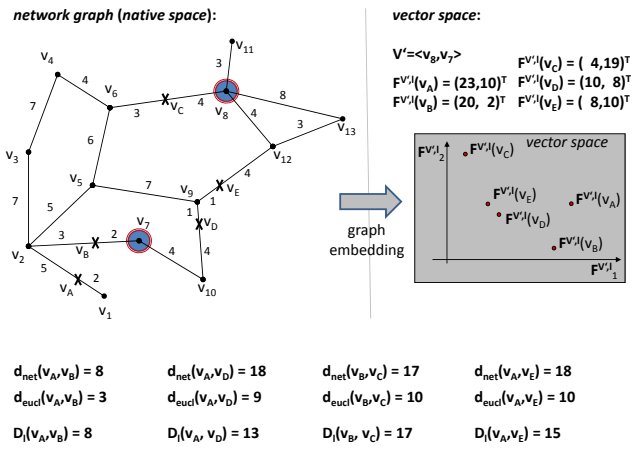


Fig. 1. Network graph embedding.

At each graph node n, the minimal cost according to each road attribute is determined and maintained in a vector. In fact, the embedding according to one reference node and all road attributes can be done by one single scan on the graph. This is done for all reference nodes, such that the overhead of the whole preprocessing step is restricted to k complete graph expansions if we assume k reference nodes.

As mentioned above the reference node embedding can be used to compute lower bounds for the network distance according to each road attribute.

Definition 5 (Network Distance Estimation): Let $\mathcal{G} = (V, E, W)$ and $F^{V',j}$ be the reference node embedding of $\mathcal{G}$ w.r.t. $V' \subset V$ according to a road attribute l. For any path $p = (v_s, \ldots, v_t)$ and any road attribute $1 \leq j \leq d$, the network distance according to l can be estimated by

$$D_l(v_s, v_t) = \max_{i=1..k} |F_i^{V',l}(v_s) - F_i^{V',l}(v_t)|.$$

In [19] it is shown that the distance $D_l(v_s, v_t)$ lower bounds the network distance according to the l-th road attribute. Thus, we can employ this embedding as lower bound approximation for A^*-search.

B. Problem Definition

After these preliminaries, we can now define the route skyline in a multi-attribute network graph as follows:

Definition 6 (Route Skyline in MAG): Given the MAG $G(V, E, W)$, the result a of route skyline query $RSQ(v_s, v_t)$ for a start node v_s and a destination node v_t is the subset of all paths $P(v_s, v_t) = \{(v_s, \ldots, v_t)\}$ for which the following condition holds:

$$RSQ(v_s, v_t) \subseteq P(v_s, v_t)$$
$$\forall\ \check{p} \in RSQ(v_s, v_t), \exists\ \Pi \in \mathbb{R}^d, \forall\ p \in P(v_s, v_t) :$$
$$Pref_\Pi(\check{p}) \leq Pref_\Pi(p)$$

In other words, a route skyline query returns the subset of all paths that offer an optimal result w.r.t. an arbitrary user preference. Thus, a user does not have to specify his preferences Π first. Instead, the query retrieves all possibly optimal paths and the user can afterwards select which of the calculated paths fits best to his or her goals. Analogously to skylines in vectors spaces, a skyline implies a domination relationship, which is formalized as follows:

Definition 7 (Route Domination in a MAG): Let $p, q \in P(v_s, v_t) = \{(v_s, \ldots, v_t)\}$ be two paths leading from node v_s to node v_t. p is called to dominate q with respect to the l-th attribute iff:

$$cost^l(p) < cost^l(q) \wedge \forall 1 \leq j \leq d \wedge j \neq i : cost^l(p) \leq cost^l(q)$$

In other words, path p dominates path q if it yields at least an improvement w.r.t. one attribute and p is at least as good as q for all other attributes. The dominance relationship is closely connected to the skyline via the following lemma.

Lemma 1: Let $p, q \in P(v_s, v_t) = \{(v_s, \ldots, v_t)\}$ be two paths leading from node v_s to node v_t and let p dominate path q. Then, the following condition holds:

$$\forall \Pi \in \mathbb{R}^d \setminus \{\vec{0}\} : Pref_\Pi(\check{p}) < Pref_\Pi(q)$$

Proof: Due to p dominating q, we know:

$$\exists 1 \leq l \leq d : cost^l(p) < cost^l(q) \wedge$$
$$\forall 1 \leq h \leq d \cap h \neq l : cost^h(p) \leq cost^h(q)$$
$$\Rightarrow \forall \Pi \in \mathbb{R}^d \setminus \{\vec{0}\} :$$
$$\exists 1 \leq l \leq d : \pi_l \cdot cost^l(p) < \pi_l \cdot cost^l(q)$$
$$\wedge \forall 1 \leq h \leq d \wedge h \neq l : \pi_l \cdot cost^l(p) \leq \pi_l \cdot cost^l(q)$$

Since $Pref_\Pi(p)$ is a weighted sum where each addend $\pi_h \cdot cost^h(p)$ is smaller than $\pi_h \cdot cost^h(q)$ and there exists at least on addend l with $\pi_l \cdot cost^l(p) < \pi_l \cdot cost^l(q)$, we can conclude that $Pref_\Pi(\check{p}) < Pref_\Pi(q)$ for all positive preference vectors Π. ■

C. Requirements for Calculating Route Skylines

Compared to established solutions processing skyline queries on databases of feature vectors, our new approach has to handle several problems. First of all, the set of all possible paths between two nodes v_s, v_t cannot be assumed to be previously known. Instead, the information is implicitly stored in the MAG and the paths have to be derived before determining their costs. Thus, a naive solution could be to calculate all paths from v_s to v_t and afterwards to sort out all dominated ones. However, the number of possible paths is increasing exponentially with network distance, i.e. the minimum number of edges between two nodes. Another problem making the approach of materializing all paths unattractive is that the number of possible start and destination nodes can be rather large. Thus, unlike in databases of feature vectors where there is only one skyline for one database, a MAG allows a large amount of different skylines corresponding to all possible start and destination nodes. As a result, precomputing the set of all paths for all possible start and destination nodes would obviously cause an enormous amount of data and thus, this brute force approach is infeasible for graphs exceeding

a certain size. In databases of feature vectors, there exists a variety of solutions that significantly speeds up skyline queries. Since the feature vectors are all completely known, methods like [26] can organize their objects in efficient data structures like the R-Tree [11] or the R^*-tree. In our setting, employing efficient index structures is not possible because the objects cannot be assumed to be materialized. Thus, without knowing the objects first, building up an index structure is not possible. As a result, the existing methods for skyline computation on feature vectors are not applicable to our setting. Due to the possibly large number of paths connecting two nodes in a MAG, an efficient solution has to prevent the calculation of dominated paths as early as possible and thus, pruning has to be done before a path is completely explored.

IV. Efficient Route Skyline Computation

In this section, we propose two algorithms for computing the route skyline between a start node and a destination node. The key points of both algorithms are the pruning strategies used to prune the search space during the graph traversal.

The principal of our first pruning criterion is the dominance relationship between routes as defined in Section III-B (cf. Definition 7). Obviously, all routes between a given start node v_s to a destination node v_t that are dominated by another route from v_s to v_t can be pruned. This can be illustrated as follows: Let $\mathcal{G}$ be a MAG with d road attributes, i.e. d-dimensional edge weights. Let us now consider the space which is defined by the domains of the edge attributes used in our graph $\mathcal{G}$. In this space which we call *route attribute space* (*RAS*), each path p in $\mathcal{G}$ can be represented by a d-dimensional point. The point in the RAS corresponds to a path (route) p in $\mathcal{G}$ and is represented by the attribute vector $p.attr[]$. In consideration of the route attribute space, the route skyline relates to the skyline based on the set of points in the *RAS* that correspond to all possible routes between a given start node and a given destination. When building the skyline in *RAS*, the usual pruning criterions can be applied, i.e. a route r is dominated by another route s, iff the attribute vector of r is dominated by the attribute vector of s in the *RAS*. In the example shown in Figure 2, all routes in the upper right space of route d which we call *pruning space* are dominated by d and, thus, can be pruned from the list of skyline candidates.

A. Basic Route Skyline Computation (BRSC)

First of all, we introduce a basic route skyline algorithm that is based on pruning the search space during the A^*-search by means of forward estimation.

1) Pruning criterion I: Pruning Based on Forward Estimation: In the following, we propose a pruning criterion that allows us to prune routes already during the route exploration. The basic idea of this strategy is to apply the forward cost estimator of the A^*-search for a partially explored route $p = (v_s, \ldots, v_i)$ (sub-route) in order to estimate the attribute vector of a (completely explored) route $\hat{p} = (v_s, \ldots, v_i, \ldots, v_t)$ that contains the sub-route p. In particular, though we don't know the exact attributes of an incompletely explored route, we can exploit the lower bounding estimation of the route attributes provided by the forward cost estimation based on our graph embedding method.

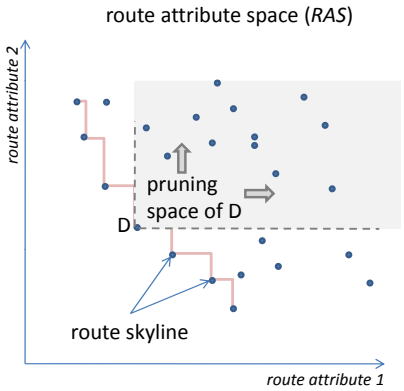


Fig. 2. Example: Route skyline in route attribute space (*RAS*).

Given the MAG $G(V, E, W)$ and the set of reference node embedding $F^{V',l}$ of $\mathcal{G}$ w.r.t. $V' \subset V$ according to each road attribute $l \in \{1, \ldots, d\}$. Let $p = (v_s, \ldots, v_i, \ldots, v_t) \in P(v_s, v_t)$ be a route from node v_s to node v_t, whereas node v_i is considered to be an intermediate node on the route p. Let $\check{p} = (v_s, \ldots, v_i)$ be a sub-route of p and let $\check{p}.attr[]$ be the corresponding attribute vector. Now, we define the attribute vector $\check{p}.lb[]$ of sub-route $\check{p}$ as follows:

$$\check{p}.lb[] = \check{p}.attr[] + (D_1(v_i, v_t), \ldots, D_d(v_i, v_t))^T.$$

where the network distance estimation $D_l(v_i, v_t)$ lower bounds the network distance between the nodes v_i and v_t according to the road attribute l by means of the reference node embedding (Definition 5). Due to the lower bounding property of the network distance estimation $D_l()$, the attribute vector $\check{p}.lb[]$ lower bounds the attribute vector of route p, i.e.

$$\forall l \in \{1, \ldots, d\} : p.attr[l] \geq \check{p}.lb[l].$$

This can be used to formulate a pruning criterion for the route exploration task.

Lemma 2: Given the MAG $G(V, E, W)$, the *route attribute space* (*RAS*) and let $\check{p} = (v_s, \ldots, v_i)$ be a (yet partially explored) sub-route. If the attribute vector $\check{p}.lb[]$ in RAS is dominated by an attribute vector of another route $p \in P(v_s, v_t)$, then each route s_i having $\check{p}$ as sub-route cannot belong to the route skyline $RSQ(v_s, v_t)$.

Proof: Let $p \in P(v_s, v_t)$, $p \neq \check{p}$ be a route from v_s to v_t, such that $p.attr[] \in$ *RAS* dominates the attribute vector $\check{p}.lb[]$. Since, $\check{p}.lb[]$ lower bounds the attribute vectors of all routes s_i starting with the sub-route $\check{p}$, i.e.

$$\forall l \in \{1, \ldots, d\} : s_i.attr[l] \geq \check{p}.lb[l]$$

$\check{p}.lb[]$ dominates all routes $s_i \in P(v_s, v_t)$ starting with sub-route $\check{p}$. As a consequence, the attribute vector $p.attr[]$ dominates all routes starting with the sub-route $\check{p}$. ■

Due to the above lemma, we do not need to explore a route $\check{p} = (v_s, \ldots, v_i)$ further, if $\check{p}.lb[]$ is dominated by the attribute

```
BRSC(Node v_s, Node v_t, Emb_MAGraph G)
(01)  Q_cand = ∅; // priority queue with sub-routes sorted in ascending order according to a given score function
(02)  S_route = ∅; // list of skyline routes
(03)  insert path p_0 = (v_s) into Q_cand;
(04)  WHILE Q_cand not empty DO
(05)    p = Q_cand.fetchTop(); // fetch next path (sub-route) from the queue.
(06)    IF p.lastNode = v_t THEN // route completed
(07)      IF p is not dominated by a route in S_route THEN
(08)        S_route.insert(p); // path p must be a skyline route
(09)        remove all routes in S_route that are dominated by p; // update route skyline
(10)      END IF;
(11)    ELSE; // route is not completed, thus p must be further expanded
(12)      // Pruning based on forward estimation (Pruning Criterion I)
(13)      compute attribute vector p.lb[]; //lower bounding cost estimations for each path attribute
(14)      IF p.lb[] is not dominated by any route in S_route THEN
(15)        vec_path = p.expand(); // expand actual path p by one hop (in each direction).
(16)        remove sub-routes in vec_path that include cycles;
(17)        insert sub-routes in vec_path into Q_cand.
(18)      END IF;
(19)    END IF;
(20)  END WHILE;
(21)  report S_route;
```

Fig. 3. Pseudocode of the Basic Route Skyline Computation (BRSC).

vector of another route $p \in P(v_s, v_t)$. Here, we can apply each route $p \in P(v_s, v_t)$ that has been found in previous iterations. Promising candidates are those routes stored in the current global route skyline $RSQ(v_s, v_t)$.

2) Route Skyline Algorithm I: The pseudo-code of our basic algorithm for the computation of a route skyline is shown in Figure 3. Our algorithm requires a priority queue Q_{cand} of paths (sub-routes) and a list S_{route} for organizing the skyline route candidates. The priority queue Q_{cand} is sorted in ascending order w.r.t. a given preference function. The list S_{route} organizes all (completely explored) routes $p \in P(v_s, v_t)$ that are candidates for the route skyline, i.e. that are not dominated by another route $p' \in P(v_s, v_t)$ explored so far.

The algorithm is called with a given start node v_s, a destination node v_t and the embedded MAG graph. It initiates an A^*-search applied on the embedded MAG graph starting with the start node v_s. The priority queue Q_{cand} is initialized with the pseudo-path p only consisting of the node v_s.

While Q_{cand} is not empty, we fetch the top path (sub-route) p from Q_{cand} and check whether the sub-route p has reached the destination node v_t. Let us first assume that p has reached v_t, i.e. p is a completely explored route between v_s and v_t and we can update the global skyline maintained in S_{route}. If p is dominated by any route in S_{route}, then p can be pruned. Otherwise, p is inserted into S_{route}. If p dominates an entry p' in S_{route}, then the entry p' can be removed from S_{route}.

Next we assume that p has not reached the goal v_t yet. Then, we have to check whether p can be extended into a skyline route. Thereby, we utilize the pruning criterion based on forward estimation. We compute for the sub-route p the lower bounding attribute vector

$$p.lb[] = p.attr[] + (D_1(v_i, v_t), \ldots, D_d(v_i, v_t))^T,$$

that estimates the attribute vector of all routes from v_s to v_t containing p as sub-route. Now, we check whether $p.lb[]$ is dominated by any (already completely explored) route in S_{route}. If this holds, we skip the further extensions of p w.r.t. pruning criterion I (cf. Section IV-B.1). Otherwise, p is further explored by expanding it by one hop in each direction by avoiding cycles, i.e. each node in p only occurs just once. After expanding p, the resulting sub-routes are inserted into the priority queue Q_{cand}. Though this algorithms is capable to calculate the route skyline between arbitrary nodes in a MAG, it has shortcomings w.r.t. efficiency and memory usage. The major drawback of this simple approach is that it has to keep large numbers of paths in Q_{cand} because if a path is rather short it is hard to decide whether it can be extended into a skyline route yet. Furthermore, to avoid an unnecessary further growth of Q_{cand} it is advisable to check each path for cycles before inserting it into the queue. Finally, the pruning criterion only works well if S_{route} already contains reasonable candidates.

B. Advanced Route Skyline Computation (ARSC)

To cope with the problems of BRSC named above, we now propose an advanced solution based on a novel and very effective pruning strategy. Like the basic pruning strategy introduced above, the following strategy allows us to determine for a sub-route $p \in \mathcal{G}$, i.e. a route that is not completely expanded, whether its further expansion would lead to a skyline route.

1) Pruning criterion II: Pruning Based on Sub-Route Skyline Criterion: The following lemma can be used to exclude partially expanded paths from further expansion tasks.

Lemma 3: Given the MAG $G(V, E, W)$ and the result set of a route skyline query $RSQ(v_s, v_t)$ for a start node v_s and a destination node v_t. Let the route $p = (v_s, \ldots, v_i, \ldots, v_t) \in RSQ(v_s, v_t)$ be any skyline route, where the node v_i is any intermediate node on the route p. Then, each sub-route $\breve{p} = (v_s, \ldots, v_i)$ of p must belong to the route skyline $RSQ(v_s, v_i)$

```
ARSC(Node v_s, Node v_t, Emb_MAGraph G)
(01)  Q_node = ∅; // queue with nodes composing skyline candidates sorted in ascending order according to a given score function
(02)  S_route = ∅; // list of skyline routes
(03)  insert v_s into Q_node;
(04)  WHILE Q_node not empty DO
(05)    v_i = Q_node.fetchTop();
(06)    FOR EACH p ∈ v_i.SubRouteSkyline DO
(07)      // Pruning based on forward estimation (Pruning Criterion I)
(08)      compute attribute vector p.lb[]; //lower bounding cost estimations for each path attribute
(09)      IF p.lb[] is dominated by any route in S_route THEN
(10)        remove p from v_i.SubRouteSkyline;
(11)      ELSE
(12)        IF p.flag = not processed THEN // if sub-route p is not processed yet.
(13)        vec_path = p.expand(); // expand actual path p by one hop (in each direction).
(14)        FOR EACH p' in vec_path DO
(15)          set p'.flag = processed; // mark sub-route p' as processed
(16)          IF p'.lastNode = v_t THEN // route completed
(17)            IF p' is not dominated by a route in S_route THEN
(18)              S_route.insert(p'); // path must be a skyline route
(19)              remove all routes in S_route that are dominated by p'; // update route skyline
(20)            END IF;
(21)          ELSE // route is not completed and path must be further expanded
(22)            // Pruning based on sub-route skyline criterion (Pruning Criterion II)
(23)            v_next = p'.end_node; // access end node of the actual path
(24)            IF p' is not dominated by v_next.SubRouteSkyline THEN
(25)              insert p' into v_next.SubRouteSkyline;
(26)              remove all sub-routes in v_next.SubRouteSkyline that are dominated by p';
(27)            END IF;
(28)            update Q_node;
(29)          END IF;
(30)        END FOR;
(31)      END IF;
(32)    END FOR;
(33)  END WHILE;
(34)  report S_route;
```

Fig. 4. Pseudocode of the Advanced Route Skyline Computation (ARSC).

called *sub-route skyline* among the sub-routes $P(v_s, v_i)$ from node v_s to node v_i.

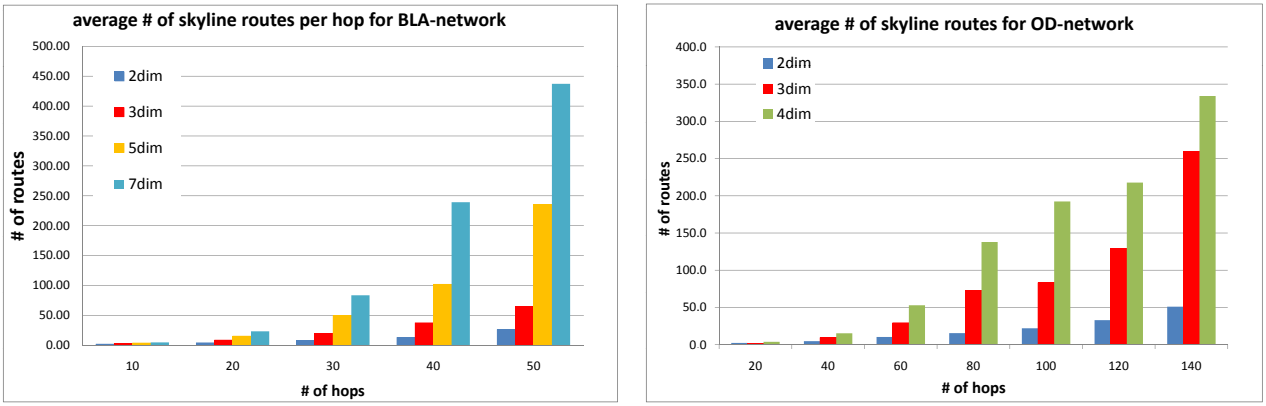
Proof: The lemma above can be easily proofed by contradiction. Assume, we have given a route $p = (v_s, \ldots, v_i, \ldots, v_t)$ that is not dominated by another route in $P(v_s, v_t)$ i.e. $p \in RSQ(v_s, v_i)$. Furthermore, let the route $p' = (v_s, \ldots, v_i)$ be a sub-route of p. Let us now assume that there is another route $p'' = (v_s, \ldots, v_i)$, $p' \neq p''$, such that p'' dominates p'. Then, there must exist a route $p^* \in P(v_s, \ldots, v_i, \ldots, v_t)$ having p'' as sub-route, that dominates p which is a contradiction to the initial assumption. ■

Let us assume that the search for the route skyline $RSQ(v_s, v_t)$ for a start node v_s and a destination node v_t has currently explored the path $\check{p} = (v_s, \ldots, v_i)$. Furthermore, let us assume that we can exclude $\check{p}$ from the route skyline $RSQ(v_s, v_i)$. Then, we know that we do not need to resume the expansion of $\check{p}$ because, according to Lemma 3, it cannot be the sub-route of a skyline route w.r.t. $RSQ(v_s, v_t)$. This pruning criterion is called *sub-route skyline pruning*.

Now we are left with the task to detect for each sub-path $\check{p} = (v_s, \ldots, v_i)$ which is built during the search of the route skyline $RSQ(v_s, v_t)$ whether $\check{p}$ can be excluded from the route skyline $RSQ(v_s, v_i)$. It suffices to check whether $\check{p}$ is dominated by any other path $\tilde{p} \in P(v_s, v_i)$ from node v_s to node v_i. However, the generation of the complete set $P(v_s, v_i)$ of paths is expensive. Instead, we propose to issue the dominance check only among pathes that have been built during the route skyline search process. For each path $p = (v_s, \ldots, v_i)$ which is explored during the graph expansion, we compute its attribute vector $p.attr[]$ and store this vector with the path p in a list $v_i.L$ which is assigned to v_i. Note that before inserting the an entry p in the list $v_i.L$, we check whether $v_i.L$ contains entries already dominating p. Only if p is not dominated by another entry in $v_i.L$, it is inserted with its attribute vector into the list.

2) Route Skyline Algorithm II: The pseudo-code of our advanced algorithm for the computation of a route skyline is shown in Figure 4. In contrast to the basic algorithm, the advanced algorithm requires an updatable priority queue Q_{node} of nodes. Each node n maintained in the priority queue Q_{node} stores its own sub-route skyline in a list $n.SubRouteSkyline$. In particular, this list maintains a skyline of all already explored sub-routes ending at node n such that the sub-routes in $n.SubRouteSkyline$ do not dominate each other. The advantages of the organization of nodes within the priority queue instead of paths (sub-routes) are the following:

- The queue is much smaller because each node is stored only once. As a consequence the update overhead of the queue is much smaller than the variant maintaining single paths.
- The main advantage of organizing nodes within the queue is that the sub-route skyline assigned to nodes suffices to apply the *sub-route skyline pruning*. We do not need to directly remove objects from the priority queue to do this pruning. Instead pruning is done locally at each node.
- The new sub-route skyline pruning automatically prevents



(a) Skyline sizes on BLA

(b) Skyline sizes on OD

Fig. 5. Average number of skyline route w.r.t. binned network distance in hops.

the path expansion from building cycles because a cyclic path cannot be a member of a local skyline.

The updatable priority queue Q_{node} is sorted in ascending order according to a given score function such that node v_i precedes node v_j, iff there is at least one sub-route in $v_i.SubRouteSkyline$ which has a higher score (preference) than each sub-route in $v_j.SubRouteSkyline$.

Equally to the basic algorithm, we additionally maintain a global list S_{route} organizing all (completely explored) routes $p \in P(v_s, v_t)$ that are candidates for the route skyline, i.e. that are not dominated by another route $p' \in P(v_s, v_t)$ explored so far. The algorithm is called with a given start node v_s, a destination node v_t and the embedded MAG graph. Initially, the algorithm inserts the start node v_s into the priority queue Q_{node}. Note that the sub-route skyline of this node contains only one sub-route that consists only the node v_s. While Q_{node} is not empty, we fetch the top node v_i from Q_{node}. For each sub-route p in $v_i.SubRouteSkyline$, we check whether p can lead to a skyline route by further route exploration steps. Thereby, we utilize the pruning criterion based on forward estimation (pruning criterion I), similar to the basic algorithm. For the sub-route p, we compute the lower bounding attribute vector

$$p.lb[] = p.attr[] + (D_1(v_i, goal), \ldots, D_d(v_i, goal))^T,$$

used to estimate the attribute vector of all routes from v_s to v_t containing p as sub-route. Now, we check whether $p.lb[]$ is dominated by any (already completely explored) route in S_{route}. If this is the case, we can prune the further exploration of p according to our pruning criterion I. All sub-routes $p \in v_i.SubRouteSkyline$ that were not processed in previous iterations are expanded by one hop in each direction. To recognize already explored sub-routes, we mark each already expanded sub-route with a *processed* flag.

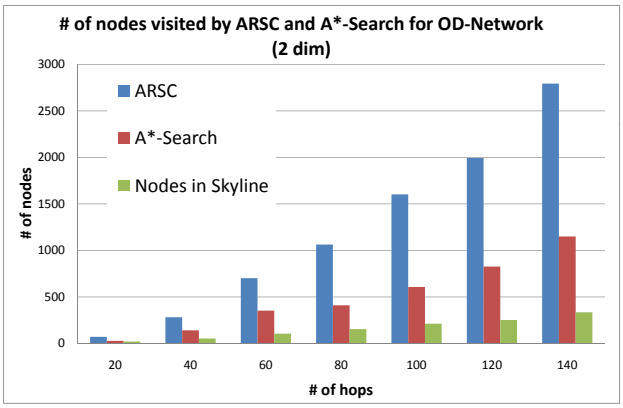
After expanding a sub-route p by one hop ending at node v_j, we first have to check whether we reached the goal, i.e. whether $v_j = v_t$. Let us first assume that we reach the goal. Then we have a further completely explored route between v_s and v_t and we can update the global skyline maintained in S_{route}. If p is dominated by any route in S_{route}, then p can be pruned. Otherwise, p is inserted into S_{route}. If p dominates an entry in S_{route}, then the entry can be removed from S_{route}.

Next, we assume that we do not reach the goal by the last route expansion. Then, we apply the sub-route skyline pruning (pruning criterion II) before updating the sub-route skyline assigned to v_j. Before inserting p into $v_j.SubRouteSkyline$, we test for each sub-route $\check{p} \in v_j.SubRouteSkyline$ whether p is dominated by $\check{p}$. If this is the case, we can prune p and do not need to insert p into the sub-route skyline. In addition, we check whether $\check{p}$ is dominated by p in order to prune the skyline route candidates maintained in v_j. If p is not dominated by any sub-route in $v_j.SubRouteSkyline$, it is inserted into $v_j.SubRouteSkyline$. Finally, the priority queue Q_{node} has to be updated, if the update of $v_j.SubRouteSkyline$ has caused a new smaller value w.r.t. the score function.

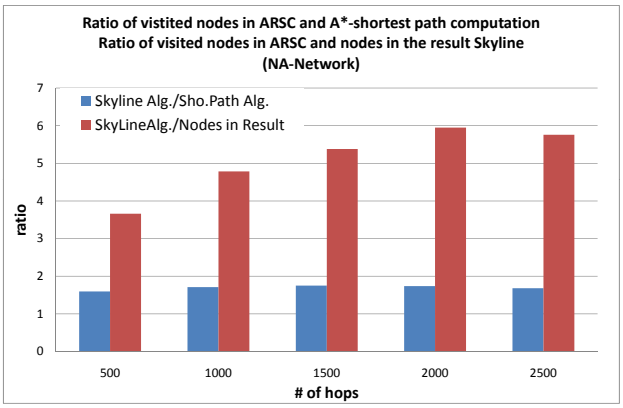
V. Experiments

In this section, we analyze the behavior of our new algorithms BRSC and ARSC. Since computing route skylines is a new problem, we additionally present results which rather describe the problem complexity than the particular performance of the algorithms.

We used three different road network graphs having different sizes. The first network graph (OD) corresponds to the road network of the city of Oldenburg, Germany, which is a well-known benchmark road network frequently used by diverse query processing approaches. It contains about 7,036 road segments and 6,105 intersection nodes. The second network graph (BLA) is a smaller excerpt of the OD graph containing only 679 segments and 532 intersections. We use this rather small graph to compare to BRSC because this simple approach required too much main memory for larger graphs like OD. To demonstrate that our second approach ARSC is capable to calculate route skylines even in large network environments, we tested this algorithm on the third graph (NA) modeling major streets in North America. This rather large network contains 179,178 road segments and 175,812 nodes. As first optimization criterion or dimension, we used the spatial length



(a) Overhead of visited nodes in OD



(b) Relative overhead of visited nodes in NA

Fig. 6. Overhead of ARSC computation w.r.t. the nodes in the result skyline and shortest path computation with A*-Search.

of each road segment. Since we did not have any data containing additional criteria, we added random values following a uniform distribution to generate the additionally required attributes. Let us note that the weight vector of a path will contain positively correlated values with increasing path length even if the edge weights are uncorrelated. Since we allow only positive weights extending a path means increasing all attributes.

To speed up A^*-Search, we constructed a reference point embedding for each graph. Due to the varying size of the graphs, we used a different number of randomly selected reference points for each graph. For BLA, we emloyed two reference points, for OD, we employed 10 reference points and for NA, we employed 50 reference points. As a default for our preference function, we assigned equal weights to each criterion. To categorize the network distance between the starting node and the destination node, we employed the number of hops on the shortest path w.r.t. this uniformly distributed preference function. Our experiments on the BLA network are averaged over 2500 random skyline queries and our experiments on the larger networks OD and NA are averaged over 400 arbitrary example queries. The results were binned w.r.t. the number of hops to measure the dependency of the problem on the distance between the query points. To avoid insignificant results, we made sure that a comparable number of examples was available for each bin. All experiments were performed on a Laptop having a 2.4 GHz Intel(R)Core(TM)2 DUO processor and 2 Gb Ram. All algorithms were implemented in JAVA 1.6.

In our first experiment, displayed in figure 5, we describe the number of found skyline routes w.r.t. an increasing network distance (number of hops) and various dimensionalities for the BLA and the OD network. For paths having a rather small number of hops on their preference route, the number of skyline routes is quite comparable for varying dimensionalities. However, for larger distances, employing more dimensions causes a super linear growth in the number of result routes. For example, considering the last bin of figure 5(a), it can be observed that the average number of skyline routes increases

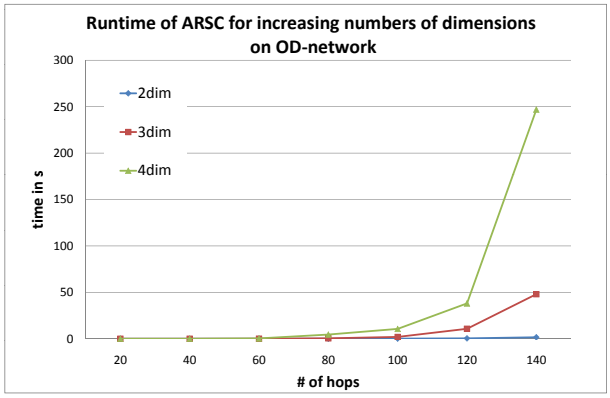
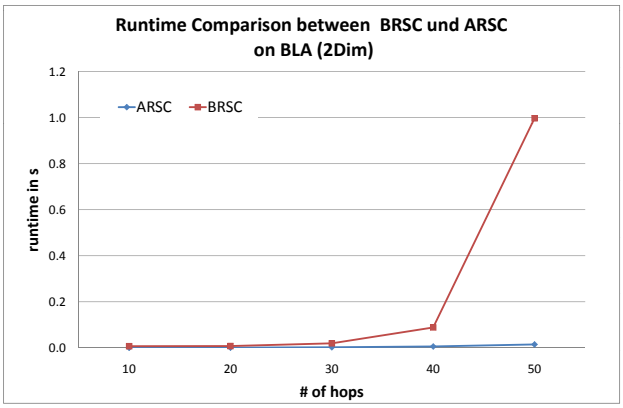


Fig. 7. Comparison of runtimes for several dimensionalities on OD.

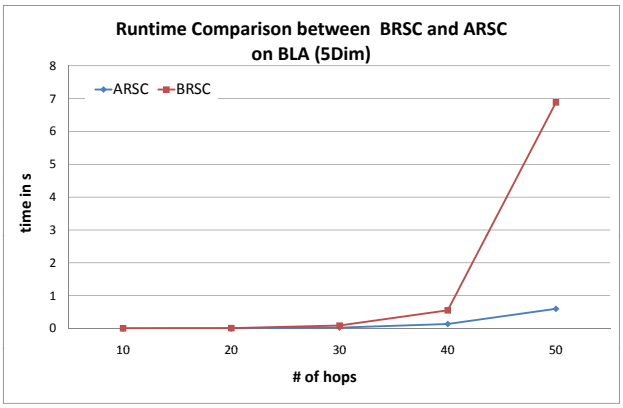
very strongly with the dimension. For 7 dimensions the skyline contained an average of 440 pareto optimal routes, even for the rather small BLA network. To sum up, the number of skyline objects strongly increases with the number of attributes making the usability of the skyline very limited.

Another interesting result is that the number of skyline routes for lower dimensionalities does not increase very strongly with the number of hops. Even for very distant nodes the number of optimal paths does not exceed 50 on the OD network. For the NA network, the average number of routes in the skyline for two dimensions is 155 for the largest category of paths having more than 2500 hops. Thus, the growth w.r.t. the network distance is still manageable even in large graphs.

In our next set of experiments, we examined the overhead of computing a route skyline compared to determining the shortest path employing A^*-search on the same embedding and using the default preference function. Furthermore, we monitored the number of different nodes that are part of any route within the skyline which is a natural lower bound for the nodes a skyline algorithm needs to visit. In figure 6(a), we display the total number of visited nodes for the OD network w.r.t. two dimensions. It can be observed that all three measures increase linearly with the number of hops and that



(a) Runtime comparison for 2 dim



(b) Runtime comparison for 5 dim

Fig. 8. Comparison of runtimes of BRSC and ARSC on the BLA networks for 2/5 dimensions.

our proposed ARSC algorithm visits in average 2.5 times the number of nodes of the corresponding A^*-search. In figure 6(b), we describe the result of a similar experiment for the NA network w.r.t. two dimensions in a slightly different way. Instead of considering the total number of nodes being visited, we display the ratio between the number of nodes visited by ARSC and A^*-search. Furthermore, we depict the ratio of nodes being visited by ARSC and the number of different nodes being contained in any skyline path. As can be seen ARSC visits about 1.8 times more nodes for all considered distance categories. Thus, the additional cost for computing the complete skyline grows with a constant rate compared to shortest path computation. Another statement that can be derived from the figure, is that the overhead of visited nodes that are not part of any skyline route is not strongly increasing with the network distance. To conclude, calculating a route skyline does not cause a super linear overhead compared to shortest path computation.

In figure 7, we examine the runtime behavior of ARSC w.r.t. the binned network distance in the OD network for 2, 3 and 4 dimensions. As expected, ARSC is capable to efficiently compute the route skyline in the majority of settings. Only for large network distances and dimensionalities larger than 2 the runtime is significantly increasing. Considering the results described in figure 5 this effect is easy to explain. Since in these settings the number of skyline routes is much larger, the algorithms need to explore a larger number of paths to find all skyline routes and thus, the runtime has to increase as well. Due to space limitations, we do not display a dedicated figure for the NA and BLA networks. In general, the performance of ARSC was quite comparable on both other graphs. On the average, it took ARSC 41 seconds to calculate a route skyline of an arbitrary distance for the NA graph and less than 5 ms for the BLA graph. Considering that ordinary shortest path computation in networks like NA are usually solved using hierarchical routing methods, the result is quite promising.

In our next experiment, we compare the runtime of BRSC and ARSC on the BLA network. Let us note that we tried to apply BRSC to larger networks like OD, but could not complete the experiments due to the enormous memory consumption of BRSC. Thus, we could only compare both algorithms on the BLA network. Figure 8 illustrates the runtime of both algorithms for two and five dimensions. In both cases, it can be seen that BRSC still performs quite well for queries in the first three bins. However, the performance of ARSC is still superior to BRSC even in these simple cases. With increasing network distance the runtime of BRSC is strongly increasing. One reason for this effect is that BRSC cannot prune any path until its forward estimation is dominated by an already retrieved result. However, for longer distances the algorithm has to manage a large number of paths until a discriminative candidate skyline is found which is capable to prune a significant amount of candidate paths. Thus, the number of paths which has to be stored and processed becomes extraordinarily large and the runtime as well as the memory requirements exceed reasonable limits. To conclude, BRSC is only applicable to smaller road networks. However, ARSC is capable to efficiently compute route skylines even for large graphs.

In the last part of our evaluation, we consider the memory requirements of both algorithms. We measured the used heap space for this experiment using the virtual machine monitor of JDK 1.6.13. Since we could not measure the exact memory consumption of the algorithm for each query, we measured the maximum memory consumption of the test programm for multiple queries. For the small Graph BLA, the BRSC algorithm already required about 10 MB of main memory, whereas the complete memory requirement of the test program running ARSC were about 800 kB. A more dramatic difference could be observed for the OD data set. While ARSC required a maximum of about 10 MB, running BRSC already required more than 1.4 Gb which is more than the maximum heap space of the Java runtime environment on a 32bit architecture. Finally, we examined the ARSC algorithm for the NA network. For this rather large graph, we measured a memory consumption of about 390 MB which is acceptable for a graph of this size.

VI. Conclusion

In this paper, we introduce the new problem of route skyline computation in a multi-attribute graph (MAG). In a MAG the edges are labeled with a vector of positive weights. An example of such a graph is a road network for which multiple criteria like speed, the number of traffic lights, distance etc. are available. The resulting route skyline between a given starting point and a destination point contains the set of all routes which are optimal under an arbitrary linear weighting. To solve this problem, we first of all apply the concepts of skyline and domination to a network environment. In contrast to skyline computation in ordinary databases, the computation of a route skyline has to successively construct paths. Our solutions are based on a global pruning criterion excluding paths for which it can be guaranteed that they are not extendable into a member of the route skyline. Based on this criterion, we propose the algorithm BRSC. However, since BRSC has a large computational overhead when applied to larger networks, we propose a second algorithm called ARSC. This algorithm is based on a second pruning criterion requiring local pareto-optimality for each extended sub path. In our experimental evaluation, we demonstrate that ARSC is capable to efficiently compute route skylines even for large graphs containing more than 170,000 nodes and edges. For future work, we plan to further extend the scalability of our approach introducing a hierarchical method for route skyline query processing.

References

[1] R. Agrawal, S. Dar, and H. Jagadish. "Direct Transitive Closure Algorithms: Design and Performance Evaluation". *TODS*, 15(3), 1990.

[2] S. Borzsonyi, D. Kossmann, and K. Stocker. The skyline operator. In *Proceedings of the 17th International Conference on Data Engineering (ICDE), Heidelberg, Germany*, 2001.

[3] C.-Y. Chan, H. V. Jagadish, K.-L. Tan, A. Tung, and Z. Zhang. Finding k-dominant skylines in high dimensional space. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data,New York, NY, USA*, 2006.

[4] L. Chen and X. Lian. Dynamic skyline queries in metric spaces. In *EDBT '08: Proceedings of the 11th international conference on Extending database technology,Nantes, France*, 2008.

[5] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with presorting. In *Proceedings of the 19th International Conference on Data Engineering (ICDE), Bangalore, India*, 2003.

[6] T. H. Corman, C. E. Leiserson, and R. L. Riverst. *"Introduction to Algorithms"*. MIT Press, 1990.

[7] K. Deng, Y. Zhou, and H. Shen. Multi-source query processing in road networks. In *Proceedings of the 23th International Conference on Data Engineering (ICDE), Istanbul, Turkey*, 2007.

[8] E. W. Dijkstra. "A Note on Two Problems in Connection with Graphs". *Numerische Mathematik*, 1:269–271, 1959.

[9] A. V. Goldberg, H. Kaplan, and R. F. Werneck. "Reach for A*: Efficient point-to-point shortest path algorithms". In *Proc. of the 8th WS on Algorithm Engineering and Experiments (ALENEX), SIAM*, 2006.

[10] A. V. Goldberg and R. F. Werneck. "Computing Point-to-Point Shortest Paths from External Memory". In *Proc. of the 7th WS on Algorithm Engineering and Experiments (ALENEX), SIAM*, 2005.

[11] A. Guttman. R-Trees: A dynamic index structure for spatial searching. In *Proc. SIGMOD*, pages 47–57, 1984.

[12] X. Huang and C. Jensen. In-route skyline querying for location-based services. In *In Proc. of the Int. Workshop on Web and Wireless Geographical Information Systems (W2GIS), Goyang, Korea*, pages 120–135, 2004.

[13] Z. Huang, C. Jensen, H. Lu, and B. Ooi. Skyline queries against mobile lightweight devices in manets. In *Proc. of the 22th International Conference on Data Engineering (ICDE), Atlanta, GA*, 2006.

[14] Y. Ioannidis, R. Ramakrishnan, and L. Winger. "Transitive Closure Algorithms Based on Graph Traversal". *TODS*, 18(3), 1993.

[15] S. Jang and J. Yoo. Processing continuous skyline queries in road networks. In *International Symposium onComputer Science and its Applications (CSA2008)*, 2008.

[16] S. Jung and S. Pramanik. "HiTi Graph Model of Topographical Roadmaps in Navigation Systems". In *Proc. ICDE*, 1996.

[17] E. Köhler, R. H. Möhring, and H. Schilling. "Acceleration of Shortest Path and Constrained Shortest Path Computation". In *Proc. Int. WS Efficient and Experimental Algorithms (WEA'05)*, 2005.

[18] D. Kossmann, F. Ramsak, and S. Rost. Shooting stars in the sky: An online algorithm for skyline queries. In *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB), Hong Kong, China*, 2002.

[19] H.-P. Kriegel, P. Kröger, P. Kunath, M. Renz, and T. Schmidt. "Proximity Queries in Large Traffic Networks". In *Proc. 15th Int. Symposium on Advances in Geographic Information Systeme (ACM GIS'07), Seattle, WA*, 2007.

[20] R. Kung, E. Hanson, Y. Ioannidis, T. Sellis, L. Shapiro, and M. Stonebraker. "Heuristic Search in Data Base Systems". *Expert Database Systems*, 1986.

[21] C. Li, B. Ooi, A. Tung, and S-Wang. Dada: a data cube for dominant relationship analysis. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Chicago, IL*, 2006.

[22] X. Lin, Y. Yuan, W. Wang, and H. Lu. Stabbing the sky: Efficient skyline computation over sliding windows. In *Proceedings of the 21st International Conference on Data Engineering (ICDE), Tokyo, Japan*, 2005.

[23] X. LIN, Y. Yuan, Q. Zhang, and Y. Zhang. Selecting stars: the k most representitive skyline operator. In *Proceedings of the 23th International Conference on Data Engineering (ICDE), Istanbul, Turkey*, 2007.

[24] N. Linial, E. London, and Y. Rabinovich. "The geometry of graphs and some of its algorithmic applications". In *Proc. IEEE Symp. Foundations of Computer Science*, 1994.

[25] M. Morse, J. Patel, and H. V. Jagadish. Efficient skyline computation over low-cardinality domains. In *Proceedings of the 33nd International Conference on Very Large Data Bases (VLDB), Vienna, Austria*, 2007.

[26] D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progressive algorithm for skyline queries. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), San Diego, CA*, 2003.

[27] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. "Query Processing in Spatial Network Databases". In *Proc. VLDB*, 2003.

[28] J. Pei, B. Jiang, X. Lin, and Y. Yuan. Probabilistic skylines on uncertain data. In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases, Vienna, Austria*, 2007.

[29] J. Pei, W. Jin, M. Ester, and Y. Tao. Catching the best views of skyline: a semantic approach based on decisive subspaces. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases,Trondheim, Norway*, 2005.

[30] C. Shahabi, M. Kolahdouzan, and M. Sharifzadeh. "A Road Network Embedding Technique for k-Nearest Neighbor Search in Moving Object Databases". *Geoinformatica*, 7(3):255–273, 2003.

[31] K. Tan, P.-K. Eng, and B. Ooi. Efficient progressive skyline computation. In *VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases*, 2001.

[32] Y. Tao and D. Papadias. Maintaining sliding window skylines on data streams. *IEEE Trans. on Knowl. and Data Eng.*, 18(3), 2006.

[33] Y. Yuan, X. Lin, Q. Liu, W. Wang, J. Yu, and Q. Zhang. Efficient computation of the skyline cube. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases,Trondheim, Norway*, 2005.

[34] Z. Zhang, X. Guo, H.L., A. Tung, and N. Wang. Discovering strong skyline points in high dimensional spaces. In *CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management, Bremen, Germany*, 2005.