Linear Path Skylines in Multicriteria Networks

Michael Shekelyan, Gregor Jossé, Matthias Schubert

Institute for Informatics, Ludwig-Maximilians-University Munich
Oettingenstr. 67, 80538 Munich, Germany
{shekelyan,josse,schubert}@dbs.ifi.lmu.de

***Abstract*—In many graph applications, computing cost-optimal paths between two locations is an important task for routing and distance computation. Depending on the network multiple cost criteria might be of interest. Examples are travel time, energy consumption and toll fees in road networks. Path skyline queries compute the set of pareto optimal paths between two given locations. However, the number of skyline paths increases exponentially with the distance between the locations and the number of cost criteria. Thus, the result set might be too big to be of any use. In this paper, we introduce multicriteria linear path skyline queries. A linear path skyline is the subset of the conventional path skyline where the paths are optimal under a linear combination of their cost values. We argue that cost vectors being optimal with respect to a weighted sum are intuitive to understand and therefore, more interesting in many cases. We show that linear path skylines are convex hulls of an augmented solution space and propose an algorithm which utilizes this observation to efficiently compute the complete linear path skyline. To further control the size of the result set, we introduce an approximate version of our algorithm guaranteeing a certain level of optimality for each possible weighting. In our experimental evaluation, we show that our approach computes linear path skylines significantly faster than previous approaches, including those computing the complete path skyline.**

I. Introduction

In many important applications areas, data is organized as a network or graph. One of the most important tasks in networks is to compute a cost-optimal path between a start node and a target node. In spatial transportation networks such as road networks, public transportation networks or computer networks, computing cost-optimal paths is the core functionality of routing and navigation functions. In other networks, such as co-authorship or social networks computing cost-optimal paths is used to measure the distance between nodes. Depending on the given application, the cost for traversing a link has a different meaning. In road networks, the cost might represent travel time, travel distance, energy consumption or the number of traffic lights. In computer networks, costs might represent the used bandwidth and the latency between routers. When considering more than one cost criterion at a time, the cost of a link or a complete path is denoted as a vector where each dimension consists of the cost value for one cost criterion. Thus, the definition of an optimal path has to be modified. A simple way is to map the cost vector to a value by employing a monotonic combination function. However, finding a suitable way of combining cost values is often a difficult task because different types of costs might have different levels of scale. Furthermore, naming an exact weighting for each type of cost is usually rather inconvenient for a user. An alternative, to selecting a combination function, is to compute the set of pareto optimal cost vectors, i.e. all result paths being optimal under any monotonic combination function. The set of all paths between a given start and target node having a pareto optimal cost vector is called path skyline [1]. Though path skyline queries compute all potentially optimal paths, they have several practical limitations. In general, the number of pareto optimal paths might increase exponentially as a function of the distance in the network and the amount of considered cost criteria. In these cases, processing time and memory consumption even for state-of-the-art methods usually degenerate. A further drawback is that confronting a user with a large amount of alternative results is often not very helpful. Thus, it does make sense to consider which type of result paths are useful and which result paths can be omitted.

In this paper, we argue that computing results for any type of combination function might add results which are not intuitively understandable in many cases. Therefore, we reduce the result set to paths which are optimal under the most basic type of monotonic combination functions, the weighted sum or linear combination. We denote the task of computing all paths being optimal under a linear combination function as linear path skyline query. Intuitively, this corresponds to the case where a user would weight each type of cost with a percentage describing its importance. The linear path skyline is a subset of the conventional path skyline and has in most cases a much smaller cardinality. Having a smaller result set yields the opportunity to improve the efficiency of query processing. Therefore, we will present a new algorithm for computing linear path skylines for an arbitrary amount of cost criteria being considerably faster and more memory efficient than state-of-the-art algorithms for processing conventional path skylines. From a mathematic point of view, the linear path skyline corresponds to the convex hull in an augmented set of cost vectors. A simple way to compute the linear path skyline is to compute the conventional path skyline and then compute the convex hull on the resulting cost vectors. However, this approach is rather inefficient due to the large amount of paths which have to be examined while computing the much larger conventional path skyline.

In contrast, our proposed algorithm LSCH constructs the linear path skyline successively while only adding new paths which are members of the result set. To add a new cost vector, we employ single criterion shortest path searches which combine the cost vectors based on the normal vectors of the hyperplanes currently limiting the linear path skyline. To identify areas on the linear skyline where additional results might still exist, we employ multidimensional convex hull computations. To process path searches as fast as possible, we employ a query specific precomputation step to determine

lower bounds for each cost criterion. Based on these lower bounds single-criterion shortest path searches can be processed very efficiently by using A*-search. Let us note that there are already solutions for computing linear path skylines for the special case of having exactly two cost criteria [2]. However, these methods heavily exploit the mechanics unique to two-dimensional cost spaces and cannot be easily generalized to our setting considering an arbitrary amount of cost criteria. Though the linear skyline is usually much smaller than the conventional skyline, the amount of result paths might still exceed the amount of practically usable alternative paths. Thus, we present an approximative variant of our algorithm which we name ε-LSCH. The result of ε-LSCH guarantees that for any possible linear weighting there exists a result path where the combined cost value is at most $\varepsilon\%$ worse than the cost value of the optimal path w.r.t. that weighting. We will compare our new methods to a state-of-the-art method for computing conventional path skylines and a method for combinatorial optimization that is adaptable to our setting. Our experiments are run on two different types of networks. The first network is a road network of the city of Munich with up to five cost criteria that is extracted from Open Street Map. The second type of network are artificial lattice graphs allowing to simulate different problem instances and parameter settings. To conclude, the contributions of the paper are as follows:

- An introduction to linear path skyline queries and their connection to multidimensional convex hulls.
- The algorithm LSCH (Linear Skyline Convex Hull) which constructs only linear path skyline paths and computes the complete linear path skyline.
- The approximative algorithm ε-LSCH which computes a subset of paths and guarantees that for each possible weighting the results divert only by a given percentage ε from the optimal costs.

The rest of the paper is structured as follows: Section II reviews related work in the area of path skyline computation, linear skylines and multidimensional convex hulls. The general setting and the definition of our task are described in section III. Our new algorithms are presented in section IV. In section V, we demonstrate the performance of our new methods based on experiments on real and artificial network data. The paper is concluded by section VI summarizing the contributions and giving an outlook at future work.

II. Related Work

In this section, we will survey related problems to linear path skylines and convex hulls.

In the database community, the skyline operator was introduced in [3]. Multiple approaches to compute skylines in database systems on sets of cost vectors followed [4], [5], [6].

The most similar setting to the approach being presented in this paper is computing route skylines [1]. In this task, we want to find all paths between two nodes in a multicriteria network where the costs of paths are not dominated by any other path between the same two nodes. In Operations Research, the problem of finding complete path skylines is known as the Multiobjective Shortest Path problem, and surveys on existing solutions to this problem can be found in [7], [8], [9], [10]. Early on, [11] proved that the size of the path skyline may increase exponentially with the number of hops between start and target node, and that the problem therefore is NP-hard. More recently, [12] showed that the number of routes is in practice feasibly low when using strongly correlated cost criteria. The current state of the art of computing path skylines are label correcting methods [1], [13], [14] employing lower bounds. In this setting, paths which cannot be extended into result paths are pruned by two mechanisms. The first mechanism is that paths ending at an intermediate node n do not need to be extended if they are dominated by any other path beginning at the start node s and ending at the intermediate node n. The second pruning method is based on the availability of a lower bound approximation of the remaining cost for each cost criteria to the target t. Based on this lower bounds it is possible to exclude paths early by comparing them to the current set of result paths between s and t. In [1], lower bounds are provided by a Reference Node Embedding computed before any query is posed. Another approach is to compute lower bound costs individually for each query [15]. Tung and Chew [16] proposed to perform a single-source all-target Dijkstra search for each cost criterion on the graph in reverse direction to find the costs of the shortest paths from all other nodes in the graph to the target node t. We will refer to this approach as Multi-Dijkstra (MD) and employ it as the lower bound computation step in our experiments because it represents the state of the art of lower bound computations for path skyline computation [14].

The methods reviewed so far aim at computing pareto optimal paths which constitute a superset of the linear path skyline discussed in this paper. There already exist some approaches to directly compute linear path skylines. The first approaches are limited to the case of two cost criteria [17], [2], [18]. [18], [17] find all non-dominated paths in a two-phase approach. The first phase computes the set of so-called supported solutions, followed by the second phase determining the remaining results. These supported solutions are equivalent to members of the linear path skyline. In [17], the authors compared several algorithms for determining the supported solutions. [2] introduces a label correcting approach to compute linear path skylines in bicriteria networks which outperformed previous approaches. However, all of the previous approaches heavily rely on the characteristics of a two dimensional cost space as documented in [19], [20], [21].

In [22] the authors propose an algorithm called ExA for computing extreme supported nondominated (ESN) points of multiobjective mixed integer programs. Since computing all ESN points is similar to computing the linear skyline, ExA can be modified to process linear path skyline queries in the multicriteria case. For d cost criteria, ExA searches all sets of cost vectors having a cardinality of d and thus, specify a hyperplane in the solution space. Each of these sets is called a stage. When processing a stage, ExA has to solve a linear equation system to determine the normal vector of the plane. Afterwards the normal vector is used to determine whether it is necessary to compute a new solution based on the stage. A drawback of ExA is that in some settings the number of stages can dramatically increase, making the number of linear equation systems which have to be solved extremely large. In contrast, our new approach works with facets on the convex hull instead of stages. Though computing facets causes an additional step

in the algorithms, there are far less facets than stages and thus, the amount of linear equation systems which have to be solved in our new approach is considerably smaller. Since ExA is the only approach for computing linear path skyline queries in multicriteria networks so far, we compare to ExA in our experiments. A well known mathematical concept being very similar to the linear skyline of cost vectors is the convex hull. The linear path skyline corresponds to a subset of the vertices on the convex hull of cost vectors. We will formally discuss the connection of the linear skyline and the convex hull when discussing our algorithm in section IV. Our algorithm will use multidimensional convex hull computations to prune the search space. Therefore, we give an overview of the used algorithm on multidimensional convex hull computation. In our algorithm, we employ the Beneath-and-Beyond approach [23] which incrementally adds new points to the convex hull. The method starts with a trivial point set spanning a d-dimensional hyperplane. Given a new point p, it is tested whether p is inside the hull. If so, the next point is examined. If p is placed outside of the hull, the algorithm removes the facets being shaded by p and computes a set of new facets. Though there exist more recent and more sophisticated methods to compute the convex hull, Beneath-and-Beyond fits very well to the requirements of our algorithm. The convex hull computation in our new method is started when a single element is added to the convex hull and returns the resulting facets containing the new point. Let us note that the setting in our algorithm can even guarantee that each added point is part of the convex hull and thus, beneath and beyond only has to compute new facets replacing an old one. Let us note that faster convex hull algorithms like quick hull [24] mainly optimize the selection of new points extending the convex hull, but do not improve the step of computing its new surface compared to Beneath-and-Beyond.

III. Problem setting

In this section, we present the formalization of concepts central to our work. First, we define the notion of a multicriteria network in order to introduce the concept of a path. Then, we formalize different concepts of skylines and the queries which compute them.

A d-dimensional multicriteria network is a directed graph $G = (V, E)$ where V denotes the set of vertices and $E \in V \times V$ denotes the set of directed edges. The dimensionality of the network refers to the number of cost functions. In a d-dimensional multicriteria network d cost functions $c_1, \ldots, c_d : E \to \mathbb{R}_{\geq 0}$ exist. For example, if G describes a three-dimensional road network, the cost functions may represent height differences, travel distances and the number of traffic lights. We denote the d-dimensional cost vector of an edge $(u, v) \in E$ by $c((u, v)) := [c_1(u, v), \ldots c_d(u, v)]^T := [(u, v)_1, \ldots, (u, v)_d]^T$. A path P is a consecutive set of edges $((s, v), \ldots, (u, t))$ which does not visit any node twice. Likewise, any path has a d-dimensional cost vector $p = (p_1, \ldots, p_d) = \sum_{(u,v)\in P} c((u, v))$ which is the componentwise sum of its edges. Note that most of the arguments presented in this paper refer to cost vectors. Let us stress that speaking of a given cost vector, we implicitly assume there exists a given path (between two nodes in a multicriteria network) with which the cost vector is associated. For reasons of brevity, however, we might omit explicitly mentioning the path which constitutes a given cost vector. When comparing different cost vectors, we always assume the corresponding paths to start and end at the same nodes within the underlying multicriteria network. When referring to multiple cost vectors, we either choose different letters (e.g. p, q) or superscript their indices (e.g. p^i). For given start and target nodes $s, t \in V$, respectively, we denote the set of all paths from s to t by $\mathcal{R}(s, t)$ or $\mathcal{R}$ is s and t are clear from the context. Throughout this paper, we assume G to be a d-dimensional multicriteria network.

In order to distinguish between linear skylines and skylines in the ordinary sense, we first give the definition of a conventional skyline followed by that of a linear skyline. Subsequently, these definitions are extended to path skylines.

Definition 1: Conventional Skyline
Let $\mathcal{P}$ be a set of points in a d-dimensional vector space. Then $x \in \mathcal{P}$ is said to *dominate* $y \in \mathcal{P}$, denoted as $x \prec y$, iff

$$\exists\ 1 \leq i \leq d : x_i < y_i \wedge \nexists\ 1 \leq i \leq d : x_i > y_i.$$

The set of points which are not dominated, i.e. $\{x \in \mathcal{P} \mid \nexists\ y \in \mathcal{P} : x \prec y\}$, is referred to as *conventional skyline* of $\mathcal{P}$ and denoted by $\mathcal{S}_C(\mathcal{P})$.

Definition 2: Linear Skyline
Let $\mathcal{P}$ be a set of points in a d-dimensional vector space. A subset $\mathcal{S}_L(\mathcal{P}) \subseteq \mathcal{P}$ is called *linear skyline*, iff the following two conditions hold:

(i) *Completeness:* $\forall\ 0 \neq w \in \mathbb{R}^d_{\geq 0}\ \exists\ x \in \mathcal{S}_L$ such that: $w^T x = \min w^T y$ for all $y \in \mathcal{P}$.

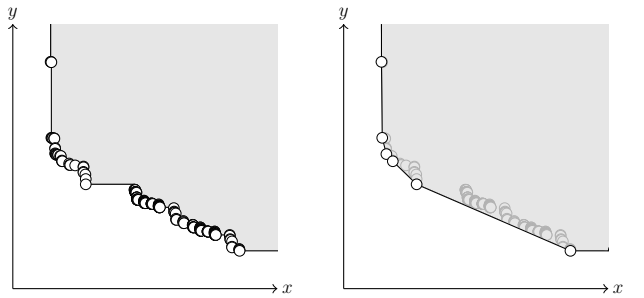
(ii) *Minimality:* $\forall\ x \in \mathcal{S}_L\ \exists\ 0 \neq w \in \mathbb{R}^d_{\geq 0}$ such that: $w^T x < w^T y$ for any $x \neq y \in \mathcal{P}$.

Now, we extend these definitions to the case of paths in multicriteria networks. The definitions extend naturally since for every path there exists a unique cost vector as defined above.

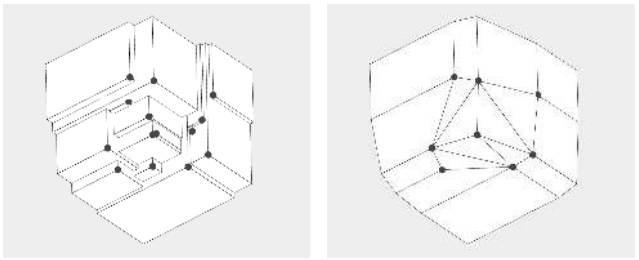
Definition 3: Conventional Path Skyline
Let $s, t \in V$ and let $\mathcal{R}(s, t)$ be the set of paths between s and t in the multicriteria network $G(V, E)$. The set of paths $\mathcal{S}_C = \mathcal{S}_C(\mathcal{R}(s, t)) = \{P^1, \ldots, P^k\} \subseteq \mathbb{R}(s, t)$ whose cost vectors $p^1, \ldots, p^k$ form the conventional (point) skyline is referred to as *conventional path skyline*.

Definition 4: Linear Path Skyline
Let $s, t \in V$ and let $\mathcal{R}(s, t)$ be the set of paths between s and t in the multicriteria network $G(V, E)$. The set of paths $\mathcal{S}_L = \mathcal{S}_L(\mathcal{R}(s, t)) = \{P^1, \ldots, P^k\} \subseteq \mathbb{R}(S, t)$ whose cost vectors $p^1, \ldots, p^k$ form the linear (point) skyline is referred to as *linear path skyline*.

Note that $\mathcal{S}_L \subseteq \mathcal{S}_C$, which is illustrated in Figure 1 on a two-dimensional and three-dimensional dataset. This is also implied by the following property: Given a point set $\mathcal{P}$, the conventional skyline is formed by those points of $\mathcal{P}$ which are optimal under some monotonic cost function. The linear skyline, in contrast, is formed by those points which are optimal under a linear combination. Since the combination functions incorporated in the definition of the linear skyline are a subset thereof, it yields the above inclusion.



(a) Conventional Skyline 2D (b) Linear Skyline 2D



(c) Conventional Skyline 3D (d) Linear Skyline 3D

Fig. 1. Comparison of the conventional and linear skylines for a 2D and 3D dataset. In 3D it is viewed from the origin of the cost space.

Also, in the definition of the linear skyline it suffices to require $w \in \mathbb{R}^d_{>0}$ instead of $w \in \mathbb{R}^d_{\geq 0}$. Since this property is of importance to later results, it is proven in the following lemma.

Lemma 1: Let $\mathcal{P}$ denote finite set of points in a d-dimensional vector space. Let $0 \neq w \in \mathbb{R}^d_{>0}$ be an arbitrary weight vector, and let $x \in \mathcal{P}$ such that $w^T x < w^T y$ for any $x \neq y \in \mathcal{P}$. Then there exists $\hat{w} \in \mathbb{R}^d_{>0}$ for which holds: $\hat{w}^T x < \hat{w}^T y$ for any $x \neq y \in \mathcal{P}$.

Proof: Let $I \subseteq \{1, \ldots, d\}$ be such that $w_i = 0$ for all $i \in I$ and $w_j > 0$ for all $j \notin I$. Let $k := |I|$ denote the number of dimensions in which w is zero. If $k = 0$, then $w \in \mathbb{R}^d_{>0}$ is already the case. Therefore, we assume $k > 0$, and by $w \neq 0$, we have $k < d$. Furthermore, let $M := \max\{y_i \mid y \in \mathcal{P}, 1 \leq i \leq d\}$ denote the maximal component of all points in $\mathcal{P}$. By assumption, we have $w^T x < w^T y$ for all $x \neq y \in \mathcal{P}$. Let $m := \min\{w^T y - w^T x \mid x \neq y \in \mathcal{P}\} > 0$. For w with k components being zero, we now construct $\hat{w} \in \mathbb{R}^d_{>0}$ for which the required condition holds. Let $w'_i := w_i$ for all $i \in I$, and let $w'_j := w_j + q$ for all $j \notin I$ where $q := m/(2kM) > 0$. Hence, $\hat{w} \in \mathbb{R}^d_{>0}$. Therefore, it remains to show that indeed $\hat{w}^T x < \hat{w}^T y$ for any $x \neq y \in \mathcal{P}$. To prove this, we use the following inequalities: $\hat{w}^T x < w^T x + qkM$ and $\hat{w}^T y > w^T y$.

$$\begin{aligned} \hat{w}^T y - \hat{w}^T x &> w^T y - \hat{w}^T x \\ &> w^T y - w^T x - qkM \\ &> m - \frac{m}{2kM} kM = \frac{2}{m} > 0 \end{aligned}$$

This proves the lemma. ∎

From the above lemma, we may derive the following equivalence in the definition of the linear skyline.

Remark 1: In Definition 2, requiring the weight vectors w in condition (i) and (ii) to be strictly positive, i.e. $w \in \mathbb{R}^d_{>0}$, yields the same result set.

Proof: Let $\mathcal{S}_L^{\geq}$ denote the linear skyline as defined by Definition 2, and let $\mathcal{S}_L^{>}$ denote the set of points which are described by the definition if $w \in \mathbb{R}^d_{>0}$ is required. Furthermore, let $\mathcal{P}$ denote set of d-dimensional cost vectors. We show: $\mathcal{S}_L^{\geq} = \mathcal{S}_L^{>}$. Obviously, $\mathcal{S}_L^{>} \subseteq \mathcal{S}_L^{\geq}$, because for $x \in \mathcal{S}_L^{>}$, there exists $0 \neq w \in \mathbb{R}^d_{>0} \subset \mathbb{R}^d_{\geq 0}$ such that $w^T x < w^T y$ for all $x \neq y \in \mathcal{P}$. Now, let $x \in \mathcal{S}_L^{\geq}$. By definition, there exists $0 \neq w \in \mathbb{R}^d_{\geq 0}$ such that $w^T x < w^T y$ for all $x \neq y \in \mathcal{P}$. Lemma 1 states that it suffices to require $0 \neq w \in \mathbb{R}^d_{>0}$. Therefore, $x \in \mathcal{S}_L^{>}$ which proves the statement. ∎

As mentioned before, it is our goal to speed up skyline computation while limiting the size of the result set. $\mathcal{S}_L$ generally holds significantly less elements than $\mathcal{S}_C$, however, we may trim the result set even further. For this purpose we introduce the notion of an ε-linear skyline.

Definition 5: ε-Linear Skyline
Let $\mathcal{P}$ be a set of points in a d-dimensional vector space, and let $\varepsilon > 0$. A subset $\mathcal{S}_\varepsilon \subseteq \mathcal{P}$ has the so-called *ε-linear skyline* property, iff the following two conditions hold:

(i) *Completeness:* $\forall\ 0 \neq w \in \mathbb{R}^d_{\geq 0}\ \exists\ x \in \mathcal{S}_L$ such that: $w^T x \leq (1+\varepsilon) \min w^T y$ for all $y \in \mathcal{P}$.

(ii) *Minimality:* $\forall\ x \in \mathcal{S}_L\ \exists\ 0 \neq w \in \mathbb{R}^d_{\geq 0}$ such that: $w^T x < w^T y$ for any $\mathcal{P} \ni y \neq x$.

As before, we extend this definition to paths in a multicriteria network:

Definition 6: Let $s, t \in V$ and let $\mathcal{R}(s,t)$ be the set of paths between s and t in the multicriteria network $G(V,E)$. Furthermore, let $\varepsilon > 0$. A set of paths $\mathcal{S}_\varepsilon = \{P^1, \ldots, P^k\} \subseteq \mathbb{R}$ whose cost vectors $p^1, \ldots, p^k$ fulfill the ε-linear skyline property is referred to as *ε-linear path skyline*.

The above definitions of ε-linear (path) skylines introduce a fault tolerance for linear (path) skylines. ε is an upper bound for the deviation of the best retrieved solution from the optimal solution. By the tightened condition (i), we get $\mathcal{S}_\varepsilon \subseteq \mathcal{S}_L$. Therefore, we have the following chain of inclusion: $\mathcal{S}_\varepsilon \subseteq \mathcal{S}_L \subseteq \mathcal{S}_C$. Note that equality is improbable and only occurs for very small skyline sizes or pathological examples.

According to these notions of a skyline, we define three different query types. Given a multicriteria network, a start and a target node therein, the *conventional skyline query* computes the conventional skyline, the *linear skyline query* computes the linear skyline and the *ε-linear skyline query* computes the corresponding ε-linear skyline for an additionally given parameter ε.

IV. Computing Linear Path Skylines

A. Preliminaries

In this section, we will define convex hulls which are inherently similar to linear skylines. This similarity will in turn be emphasized in a series of statements. As a foundation for what will follow, let us formalize the notion of convex combinations. Convex combinations are weighted averages of

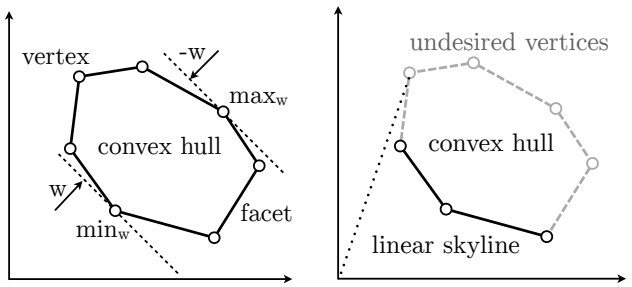


Fig. 2. The figure on the left illustrates the definition of convex hulls and the figure on the right illustrates how linear skylines relate to convex hulls.

point sets. Let $\{\alpha^{(1)}, \alpha^{(2)}, \ldots, \alpha^{(k)}\}$ be non-negative weights s.t. $\alpha^{(1)} + \alpha^{(2)} + \ldots + \alpha^{(k)} = 1$. Any $\alpha^{(1)}p^{(1)} + \alpha^{(2)}p^{(2)} + \ldots + \alpha^{(k)}p^{(k)}$ are then convex combinations of the set of points $\{p^{(1)}, p^{(2)}, \ldots, p^{(k)}\}$.

We may now introduce convex hulls in a similar way as linear skylines are defined.

Definition 7: Convex Hulls
Let $\mathcal{P} \subset \mathbb{R}^d_{\geq 0}$ be a finite set of points, i.e. cost vectors of all paths between two locations. A convex hull of $\mathcal{P}$ can be defined as the convex combination of the hull's vertices. A set of points $Conv(\mathcal{P})$ are the vertices of the convex hull of $\mathcal{P} \subset \mathbb{R}^d_{\geq 0}$, iff the following two conditions are satisfied:

(i) *Completeness:* $\forall\ 0 \neq w \in \mathbb{R}^d\ \exists\ x \in Conv(\mathcal{P})$ such that: $w^T x = \min_{y \in \mathcal{P}} w^T y$

(ii) *Minimality:* $\forall\ x \in Conv(\mathcal{P})\ \exists\ 0 \neq w \in \mathbb{R}^d$ such that for any cost vector $\mathcal{P} \ni y \neq x : w^T x < w^T y$.

The notion of a convex hull is illustrated in Figure 2. Note that the concept of linear skylines only differs from that of convex hulls in the requirement of the weight vectors being non-negative (or strictly positive by Lemma 1). This is also evident in Figure 2 (figure on the left). Another important geometric notion is needed for the following argumentation, namely the notion of facets of the convex hull:

Definition 8: Convex Hull Facets
Let $\mathcal{P} \subset \mathbb{R}^d_{\geq 0}$ be a finite set of points, i.e. cost vectors of all paths between two locations. Let $Conv(\mathcal{P})$ be the set of convex hull vertices of $\mathcal{P}$.

(i) All vertices $\{v_1, \ldots v_k\}$ of a facet lie on the same hyperplane with the normal vector $0 \neq w \in \mathbb{R}^d$, s.t. $w^T v_1 = \ldots = w^T v_k$.

(ii) $w^T v_1 = \min_{x \in Conv(\mathcal{P})} w^T x$

As we rely on a rather untypical definition of convex hulls, we shall now prove its equivalence to the conventional definition.

Lemma 2: If for a point $0 \neq x \in \mathbb{R}^d$ exists a vector $0 \neq w \in \mathbb{R}^d$ s.t. $w^T x < w^T y$ for all $x \neq y \in \mathcal{P}$, then it is not a convex combination of $\mathcal{P} \setminus x$.

Proof: The expression $w^T x$ can be interpreted as the distance from a hyperplane which goes through the origin and has the normal vector w. Convex combinations of a set of points cannot have a larger or smaller distance to the hyperplane than any of the points in the set. ■

Theorem 1: $Conv(\mathcal{P})$ is the minimal set of points s.t. all points in $\mathcal{P}$ are convex combinations of $Conv(\mathcal{P})$.

Proof: From the completeness condition in the definition follows that for each vector $0 \neq w \in \mathbb{R}^d$ there exists a convex hull vertex a s.t. $w^T a = \min_{x \in \mathcal{P}} w^T x$ and a convex hull vertex b s.t. $w^T b = \min_{x \in \mathcal{P}} -w^T x = \max_{x \in \mathcal{P}} w^T x$. From Lemma 2 therefore follows that convex hull vertices do not lie on a line segment between any two points in $\mathcal{P}$. Convex hull vertices are therefore extreme points of a finite convex set and it is well known that all points in the set are convex combinations of the extreme points.

From the minimality condition in the definition follows that for each convex hull vertex exists a hyperplane through the origin to which the vertex has either a larger or smaller distance than all other points in $\mathcal{P}$. From Lemma 2 follows that all convex hull vertices are essential. ■

B. Linear Skyline Convex Hulls

As depicted in Figure 2 (figure on the right), the linear skyline is part of the convex hull. Intuitively one could think that the linear skyline is the subset of the convex hull that is visible from the origin, but the dotted line in the figure disproves this notion. Geometrically, the linear skyline contains all points that minimize the distance to some hyperplane through the origin, but convex hull vertices can also maximize that distance. In order to get rid of the undesired vertices, the convex hull can be computed in the augmented cost vector space $\mathcal{P} \cup \text{INF}$ instead of $\mathcal{P}$. Therefore, we now define infinity points denoted by INF.

Definition 9: The set of infinity points INF is defined as follows:

$$\lim_{\omega \to \infty} \{[\omega, 0, \ldots, 0]^T, [0, \omega, \ldots, 0]^T, \ldots, [0, 0, \ldots, \omega]^T\}$$

The i-th criterion's infinity point $\text{INF}^{(i)} \in \text{INF}$ lies on the i-th criterion's axis at infinity.

$$\text{INF}^{(i)}_j = \lim_{\omega \to \infty} \begin{cases} j = i, & \omega \\ j \neq i, & 0 \end{cases}$$

The convex hull of $\mathcal{P} \cup \text{INF}$ has two facets which have exclusively infinity points as vertices. The front-infinity-facet and the back-infinity-facet. The front-infinity facet's normal vector has only positive components and the back-infinity-facet s normal vector has only negative components. The back-infinity-facet is part of any convex hull in $\mathcal{P} \cup \text{INF}$.

We call the convex hull of the augmented cost vector space $\mathcal{P} \cup \text{INF}$ the linear skyline convex hull of $\mathcal{P}$. In the following, it will be shown that the vertices of the linear skyline convex hull are simply the elements of the linear skyline in $\mathcal{P}$ and the infinity points.

Lemma 3: Let $\mathcal{P} \subset \mathbb{R}^d_{\geq 0}$ be a finite set of points, i.e. cost vectors of all paths between two locations.

$$\mathcal{S}_L(\mathcal{P}) = Conv(\mathcal{P} \cup \text{INF}) \setminus \text{INF}$$

1. Start with convex hull of infinity points.
   1.1 $P = \text{INF},\ open = \{\},\ closed = \{\}$
   1.2 Add front-infinity-facet to $open$ and back-infinity-facet to $closed$.
2. Remove an open facet F with normal vector $0 \neq w \in \mathbb{R}^d_{\geq 0}$ and vertices $V \subset P$ from $open$.
   2.1 Find x s.t. $w^T x = \min_{y \in \mathcal{P}} w^T y$.
   2.2 *If* $w^T x < \min_{y \in V} w^T y$:
      2.2.1 add x to P
   2.3 *If* $w^T x \geq \min_{y \in V} w^T y$:
      2.3.1 add F to $closed$ and go to step 4.
3. Determine facets of the convex hull and their normal vectors. Add new facets to $open$.
4. If $open \neq \{\}$ go to step 2, else return $P \setminus \text{INF}$.

Fig. 3. Proposed algorithm LSCH (Linear Skyline Convex Hull) to compute the Linear Skyline of $\mathcal{P}$.

Proof: Let positive vectors be vectors with exclusively positive components. Let non-negative vectors be vectors with exclusively non-negative components. Let negative vectors be vectors with at least one negative component. Vectors that minimize the dot product with a positive vector in $\mathcal{P}$, also minimize it in $\mathcal{P} \cup$ INF. First, from Lemma 1 follows that any vectors that minimize the dot product non-negative vectors also minimize the dot product with positive vectors. Second, the dot product of points in INF with positive vectors is clearly larger than of any points $\mathcal{P}$ with positive vectors. Vectors that minimize the dot product with a negative vector in $\mathcal{P}$, do not minimize them in $\mathcal{P} \cup$ INF. The dot product of one of the infinity points and a vector with at least one negative component is clearly smaller than of any points in $\mathcal{P}$. The rest follows from the almost equivalent definition of convex hulls and linear skylines. ■

C. Proposed Algorithm for Incremental Linear Skyline Computation

In the previous section, it was shown that the linear skyline of $\mathcal{P}$ can be obtained by computing the convex hull in $\mathcal{P} \cup$INF, but so far it was not shown how this can be used for computing linear path skyline queries. There are too many paths between two locations to create all of them. In the following, it is shown how the complete linear skyline Convex Hull can be obtained by only creating cost vectors and associated paths that are contained in $Conv(\mathcal{P} \cup \text{INF})$. The basic idea is to find new solutions in each iteration, determine the convex hull in $\mathcal{P} \cup$INF of the known solutions and search for a new solution outside of the convex hull by minimizing the dot product with the normal vectors of the hull's facets. To find a new solution which minimizes the dot product an A*-search is performed in the network which employs the normal vector for computing the cost of a path. The computation terminates once it is known that there are no more cost vectors outside the convex hull. Any solutions that are known during the computation are elements of the final result set and therefore, they can be immediately presented to the user.

The proposed algorithm to compute the linear skyline is outlined in Figure 3. An exemplary run of the algorithm for two cost criteria is shown in Figure 4. It should be noted

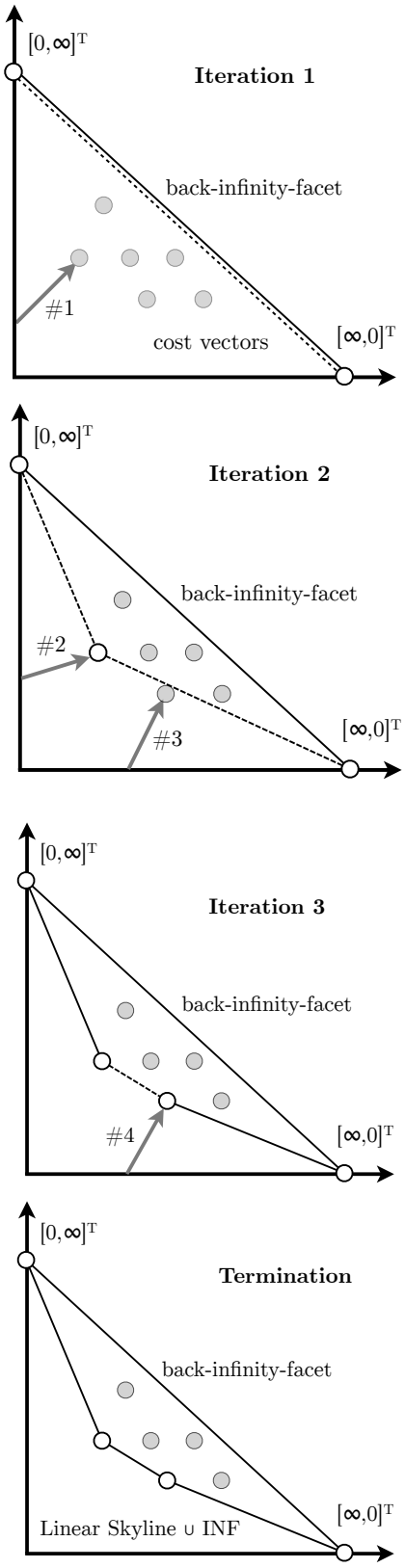


Fig. 4. Exemplary linear skyline convex hull computation which obtains a linear skyline with two cost vectors in four scalarized searches. Coordinates are schematic, because it would otherwise not be possible to display the coordinates at infinity.

that angles and distances are depicted incorrectly, in order to display the infinity points. At first the algorithm begins with the front-infinity-facet and back-infinity-facet which are both depicted as lines connecting the infinity points. Open facets are depicted as dashed lines and closed facets as solid lines. The first search minimizes the normal vector of the front-infinity-facet and finds a new solution which leads to the removal of the front-infinity-facet and the addition of two new open facets. The second and third search minimize the normal vectors of these two open facets. The second search finds an old solution and the corresponding facet is closed. The third search finds a new solution which leads to the removal of the facet and the addition of two new facets. One of these new facets has the same normal vector as a previously processed facet and is immediately closed. The fourth search finds an old solution and the corresponding facet is closed. This concludes the computation. It can be seen that instead of computing the complete convex hull of all cost vectors in $\mathcal{P}$, it only computes the linear skyline of $\mathcal{P}$.

1) Initialization: The algorithm starts with a convex hull of INF which only contains the front- and back-infinity-facet. The front-infinity-facet is added to the set of open facets and the back-infinity-facet is added to the set of closed facets. In this paper, the normal vector of the front-infinity-facet is chosen as $[1, 1, \ldots, 1]^T$ but this is not essential for the correctness of the algorithm.

The algorithm maintains a set of open facets, a set of closed facets and a list of linear skyline elements $P \subseteq \mathcal{S}_L(\mathcal{P})$. Facets are represented as a sorted integer array and vertices are represented as list indices. Infinity points can then be given negative vertex indices from $-d$ to -1 and are not part of the list, because they are not part of the linear skyline in $\mathcal{P}$.

2) Find solutions outside current Convex Hull: In the second step, the algorithm searches for further solutions outside of the current convex hull. A facet's hyperplane divides the space in two halves. One half contains the origin and the other does not. For all hyperplanes, any cost vectors inside the convex hull lie in the half that does not contain the origin. To find solutions that are outside of the convex hull, it is necessary to find cost vectors that lie on the same side as the origin for some facet's hyperplane. If such a cost vector does not exist for a facet's hyperplane, the facet is a closed facet.

First, an open facet F with the vertices V and the normal vector $0 \neq w \in \mathbb{R}^d_{\geq 0}$ is selected. How normal vectors are determined is outlined in the third step of the algorithm. Second, the cost vector x s.t. $w^T x = \min_{x \in \mathcal{P}} w^T x$ and its associated solution is computed. Third, if $w^T x$ is smaller than the minimal dot product with the facet's vertices $\min_{x \in V} w^T x$, then x lies outside the convex hull. If x lies outside the convex hull, the facet F is simply removed from the set of open facets and x is added to the list of linear skyline elements. If x does not lie outside the convex hull, the facet F is moved from the set of open facets to the set of closed facets.

Open facets of convex hulls in $\mathcal{P} \cup$INF always have normal vectors with non-negative components. By definition, vertices of facets have to lie on the same hyperplane and cost vectors in $\mathcal{P}$ always have larger dot products than cost vectors in INF for any normal vectors that contain negative components. Therefore only facets which consist solely of infinity points can have normal vectors $w \notin \mathbb{R}^d_{\geq 0}$ and the back-infinity-facet is trivially never an open facet. Minimizing the dot product with such vectors is therefore simply a search with scalarized cost objectives. For linear path skylines, scalarized searches to find cost-optimal paths between two query locations, can be performed by shortest path algorithms like Dijkstra, Bidirectional Dijkstra and A*-search. For a weight vector w and an edge's cost vector e the scalarized edge cost during shortest path search is simply $w^T e$.

If a search was previously conducted with the same weight vector w, the search would trivially yield the same result. The normal vectors of previous results should therefore be stored to prevent such redundant searches.

3) Update convex hull and determine normal vectors of facets: Finding the convex hull facets of a set of vertices is a well-known problem called facet enumeration and can be solved by most existing convex hull algorithms for an arbitrary number of dimensions. Incremental approaches like the Beneath-And-Beyond approach have the advantage, that they do not need to recompute all facets and only determine new facets. If the algorithm or its implementation does not support infinitely large numbers, it can be approximated by a very large number.

Normal vectors of facets can be determined by solving a linear equation system $Ax = b$. The matrix A has the vertices of the facets as row vectors, the vector b is any vector that has the same positive value for all components and the vector x is the resulting normal vector. When infinity points are vertices of the facet, any components where infinity points have infinitely large components have to be zero for the normal vector. The infinity points and their infinity components therefore can be excluded from the linear equation system only a $d-k \times d-k$ linear equation system has to be solved if there are k infinity point vertices.

4) Termination: If the set of open facets is not empty, the algorithm returns to the second step and tries to find cost vectors outside the new convex hull.

If the set of open facets is empty, the hyperplanes of the closed facets define the convex hull of the augmented space $\mathcal{P} \cup$ INF and the algorithm terminates. After termination, the algorithm's list of linear skyline solutions P is equal to the complete linear skyline $\mathcal{S}_L(\mathcal{P})$.

D. Linear Path Skyline Computation

In the previous section, it was shown how the linear path skyline computation can be reduced to scalarized searches. We will now describe the details about how these scalarized searches are performed. The linear path skyline computation consists of the two steps outlined in Figure 5. In the first step the shortest path distances between all nodes and the target

1. Compute shortest path costs from all nodes to the destination to prepare heuristic for subsequent A*-searches.
2. Compute Linear Path Skyline with LSCH using A*-search for scalarized searches.

Fig. 5. Proposed algorithm for linear path skyline computation.

TABLE I. COMPARISON OF KNOWN UPPER BOUNDS OF THE PROPOSED APPROACH AND KNOWN UPPER BOUNDS OF EXA.

	$k=10$	$k=50$	$k=100$	$k=250$
Worst case number of stages with ExA[25]				
$d=2$	20	100	200	500
$d=4$	130	19650	161800	2573250
$d=6$	262	2118810	75287620	7817031550
Maximal number of facets with the proposed algorithm (LSCH)				
$d=2$	20	100	200	500
$d=4$	45	1225	4950	31125
$d=6$	60	17300	152100	2511500

node are computed for all cost criteria. This yields cost vectors that can be scalarized for each of the scalarized searches. The purpose of these scalarized shortest path distances is to guide the searches towards the target and thereby speed up the multiple searches between the same two locations. In this paper, this technique is called Multi-Dijkstra, because an all-target Dijkstra search with reversed edge directions is performed for each of the d criteria [16]. In the second step, the LSCH algorithm, as outlined in Figure 3, performs scalarized searches with A*-search and uses the scalarized shortest path distances as consistent cost heuristics.

E. Correctness of the proposed algorithm for Linear Path Skyline Computation

Theorem 2: After the termination of the algorithm, P is the complete linear skyline of $\mathcal{P}$.

Proof: Let a facet with the set of vertices V and the normal vector w be a closed facet in $\mathcal{P}$, iff $\min_{x\in V} w^T x = \min_{y\in\mathcal{P}} w^T y$.

The algorithm only terminates when all facets of the current convex hull $Conv(P \cup \text{INF})$ of $P \subseteq \mathcal{P}$ are closed facets. It is a well-known property of convex hulls, that a cost vector $x \in \mathcal{P}$ lies outside the convex hull of $P \subseteq \mathcal{P}$ iff the convex hull $Conv(\mathcal{P})$ has a facet with vertices V and normal vector $0 \neq w \in \mathbb{R}^d$ s.t. $w^T x < \min_{y\in V} w^T y$. If all facets are closed, there cannot exist such a facet. Therefore, all vectors are contained in the convex hull. The rest follows from $Conv(\mathcal{P} \cup \text{INF}) \setminus \text{INF}$ being equal to the linear skyline $\mathcal{S}_L\mathcal{P}$ which is shown in Lemma 3. ∎

F. Termination and Worst Case Bounds of Proposed Algorithm

Theorem 3: Let $\mathcal{P} \subset \mathbb{R}^d_{\geq 0}$ be a finite set of points, i.e. cost vectors of all paths between two locations. Each scalarized search yields a previously unknown closed facet in $\mathcal{P}$ or a previously unknown linear skyline element.

Proof: Let $x \in \mathcal{P}$ be the solution of the scalarized search s.t. $w^T x = \min_{x\in\mathcal{P}} w^T x$. The scalarization weights $0 \neq w \in \mathbb{R}^d_{\geq 0}$ of the search are the normal vector of an open facet. Let V be the vertices of this facet. If $\min_{x\in\mathcal{P}} w^T x = \min_{x\in\mathcal{P}} v^T x$, then the facet is by definition a closed facet. If it is not the case, it follows $w^T x < \min_{x\in Conv(P\cup\text{INF})} w^T x$ because it is a closed convex hull facet in P. Therefore, it is a new solution. ∎

Table I displays the amount of facets leading to path searches in LSCH in comparison to the amount of stages being examined in ExA [25]. It can be observed that the theoretical worst case amount of facets is considerably smaller than the worst case amount of stages in ExA.

1. Start with convex hull of infinity points.
   - 1.1 $P = \text{INF},\ open = \{\},\ closed = \{\}$
   - 1.2 Add front-infinity-facet to $open$ and back-infinity-facet to $closed$.
2. Remove an open facet F with normal vector $0 \neq w \in \mathbb{R}^d_{\geq 0}$ and vertices $V \subset P$ from $open$.
   - 2.1 Find x s.t. $w^T x = \min_{y\in\mathcal{P}} w^T y$.
   - 2.2 *If* $w^T x < \frac{1}{1+\varepsilon} \min_{y\in V} w^T y$:
     - 2.2.1 add x to P
   - 2.3 *If* $w^T x \geq \frac{1}{1+\varepsilon} \min_{y\in V} w^T y$:
     - 2.3.1 add F to $closed$ and go to step 4.
3. Determine facets of the convex hull and their normal vectors. Add new facets to $open$.
4. If $open \neq \{\}$ go to step 2, else return $P \setminus \text{INF}$.

Fig. 6. Proposed algorithm (ε-LSCH) to compute the ε-Linear Skyline of $\mathcal{P}$. For $\varepsilon = 0$ this algorithm is equivalent to the proposed algorithm to compute the Linear Skyline.

Theorem 4: The maximal number of scalarized searches and convex hull updates for a linear path skyline with k solutions is with the algorithm $k + \binom{k-\lceil d/2\rceil}{k-d} + \binom{k-\lfloor d/2\rfloor}{k-d}$. The number of searches and convex hull updates is with the proposed incremental algorithm for a linear path skyline with k solutions $O(k^{\lfloor \frac{d}{2}\rfloor})$.

Proof: The algorithm performs at most one convex hull update per scalarized search. Each scalarized search yields either a new solution or a closed facet. The maximal number of searches and convex hull updates is therefore the number of solutions added with the number of closed facets. It should be noted that the vertices of the closed facets are ignored in this case and facets are only regarded as half-space representations. It is therefore irrelevant if there are multiple vertex combinations that could be used to represent one of the convex hull's intersecting half-spaces. The Upper Bound Theorem [26] states that the number of convex hull facets as a function of the number of vertices is maximal for cyclical polytopes which have $\binom{k-\lceil d/2\rceil}{k-d} + \binom{k-\lfloor d/2\rfloor}{k-d}$ facets for k vertices. The asymptotical version of the Upper Bound theorem [27] states that the number of convex hull facets as a function of the number of vertices is $O(k^{\lfloor \frac{d}{2}\rfloor})$ for k vertices. ∎

G. Computing ε-Linear Skylines

The proposed algorithm to compute ε-Linear Skylines is outlined in Figure 7. The basic idea of this approach is to check if a new solution x would lie inside the convex hull of the previous solutions multiplied by $\frac{1}{1+\varepsilon}$. If it does, previous solutions P already contain for all $0 \neq w \in \mathbb{R}^d_{\geq 0}$ a cost vector $y \in P$ s.t. $w^T y \leq (1+\varepsilon) w^T x$.

To simplify the proof of correctness, the convex hull multiplied by $\frac{1}{1+\varepsilon}$ is defined as the ε-convex hull.

Definition 10: Let $\mathcal{P} \subset \mathbb{R}^d_{\geq 0}$ be a finite set of points, i.e. cost vectors of all paths between two locations. The ε-convex hull of $\mathcal{P}$ consists of the vertices of the convex hull of $\mathcal{P}$ multiplied by $\frac{1}{1+\varepsilon}$ and is denoted by $Conv_\varepsilon(\mathcal{P})$.

$$Conv_\varepsilon(\mathcal{P}) = \left\{ \frac{1}{1+\varepsilon} x \,|\, x \in Conv(\mathcal{P}) \right\} \tag{1}$$

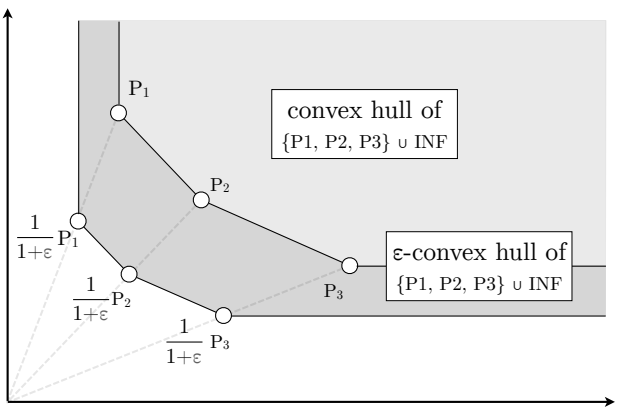


Fig. 7. Comparison of ε-convex hull with conventional convex hull in $\mathcal{P} \cup$ INF.

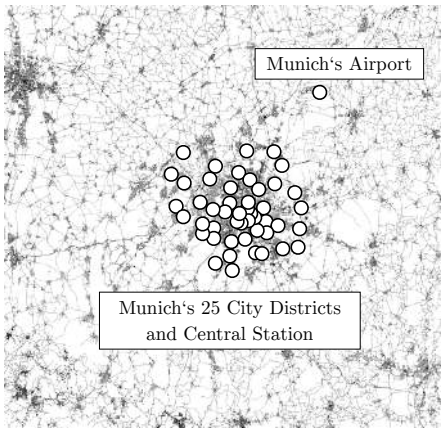


Fig. 8. Map of Munich with all selected start and end locations.

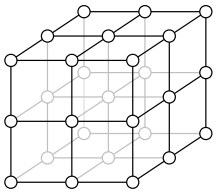


Fig. 9. $3 \times 3 \times 3$ lattice graph, routing tasks are performed between opposing corners.

Multiplying a set of points $\mathcal{P}$ by $\frac{1}{1+\varepsilon}$ and then determining the convex hull leads to the same result as first computing the convex hull of the set of points and then multiplying all convex hull vertices by $\frac{1}{1+\varepsilon}$. As can be seen in Figure 7, multiplying a set of points with a scalar simply scales all dot products by the same amount and therefore does not affect angles or distance relationships. The minima and maxima for dot products with any $0 \neq w \in \mathbb{R}^d$ are just scaled. Intuitively this is comparable to changing from coordinates in meters to coordinates in kilometers, which clearly would not affect the number of facets, which vertices belong to which facets and the facets' normal vectors.

Finally, it follows a proof for correctness.

Theorem 5: Upon termination of the algorithm, P is the complete ε-linear skyline of $\mathcal{P}$.

Proof: The algorithm only terminates when all facets of the ε-convex hull are closed facets. Analogously to the proof for the conventional LSCH algorithm in Theorem 2, all cost vectors in $\mathcal{P}$ are upon termination contained in the ε-convex hull of $\mathcal{P} \cup$ INF.

The ε-convex hull of $P \subseteq \mathcal{P}$ contains for any $0 \neq w \in \mathbb{R}^d$ the vertex $x = \min_{y \in P} w^T \frac{1}{1+\varepsilon} y$. From this follows that the convex hull contains $(1+\varepsilon)x = \min_{y \in P} w^T y$. If a cost vector $x \in \mathcal{P}$ does not lie outside the ε-convex hull of $P \subseteq \mathcal{P}$, then for each $0 \neq w \in \mathbb{R}^d$ there exists a vertex $y \in P$ of the convex hull of $P \subseteq \mathcal{P}$ s.t. $w^T y \leq \frac{1}{1+\varepsilon} w^T x$.

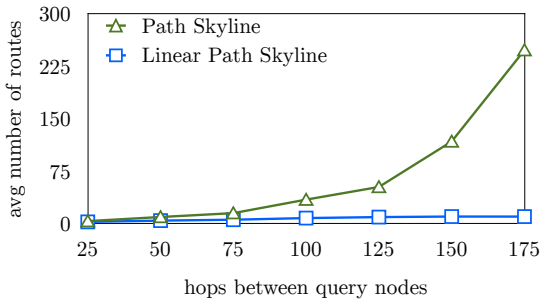
It follows that if all solutions are inside the ε-convex hull of $\mathcal{P} \cup$ INF, the vertices of the convex hull of $P \cup$ INF contain the ε-Linear Skyline. ■

V. Experimental Evaluation

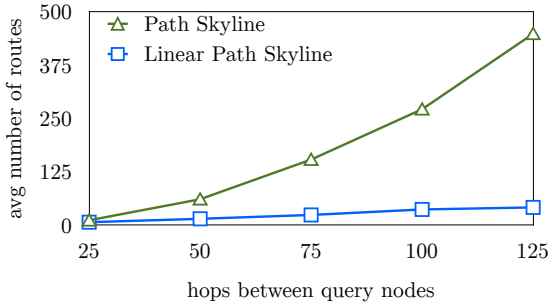
All experiments are performed on a dedicated machine with an 2.2 Ghz Opteron Dual Core processor and 64 GB RAM. All algorithms were implemented in Java 1.7 and a single core was used for each experiment. If an algorithm is not able to solve a task in less than 360 seconds the computation is aborted and it is counted as a timeout.

We tested our algorithms on two types of networks. The first type of network is the road network of Munich as derived from OpenStreetMap. This Munich graph consists of 221,465 nodes and 519,917 edges. The cost criteria utilized in the experiments are travel time, route length, number of crossings, penalized travel time and energy loss. The criterion travel time assumes travel speeds to equal the speed limits whereas the penalized travel time assumes additional 30 and 15 seconds for each crossing with and without traffic lights, respectively. The energy loss is a synthetic model roughly derived from typical battery capacities of electric cars and their respective ranges. It incorporates altitude differences in the following sense: ascent increases the energy consumption by the gained potential energy and descent reduces the energy consumption (while negative values are corrected to zero). Let us stress that the authenticity of the cost models used has absolutely no influence on the computational benefits of our algorithms. For experiments using two criteria, we employed travel time and route length. For the three criteria setting, we added the number of crossings. Finally for the five criteria settings, all five criteria were used. As sample queries we selected the centers of the 25 city districts of Munich and added the central station and the airport. Based on these start and target locations we conducted $\binom{27}{2} \cdot 2 = 702$ sample queries. Figure 8 depicts the 27 locations on the map.

The second type of graphs are three dimensional lattice graphs of varying sizes (cf. Figure 9). In other words, we generate lattices having $n \times n \times n$ nodes for $n \in \{2, 4, 6, 8\}$. Thus, the graph for $n = 6$ has $6 \cdot 6 \cdot 6 = 2016$ nodes. As attributes we artificially generated five cost values for each edge. For each graph, each pair of diagonal edges was used as a query. Thus, for each graph, we tested a total of eight queries.

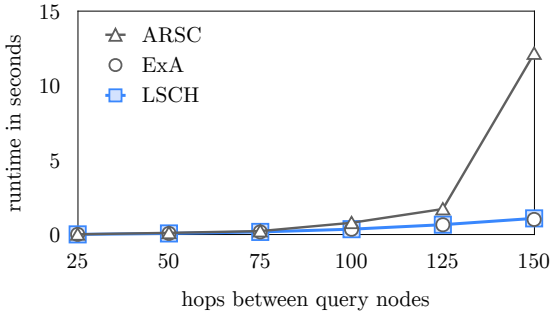


(a) 3D

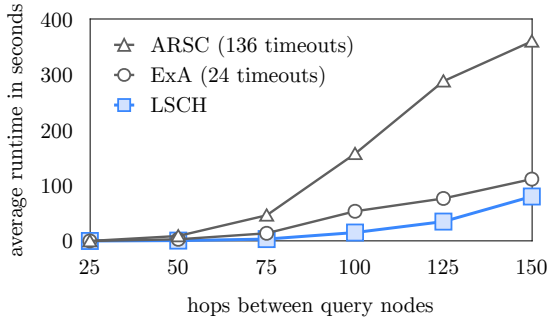


(b) 5D

Fig. 10. Comparison of the number of skyline paths for conventional and linear path skyline for 3 and 5 cost criteria in the Munich setting.



(a) 3D



(b) 5D

Fig. 11. Runtime comparison between ARSC, ExA and LSCH in the Munich setting.

We compare our new algorithms LSCH and ε-LSCH with the ExA algorithm proposed in [22]. Since ExA is designed for multiobjective linear programs instead of linear path skyline queries, we incorporated the A*-Search for computing shortest paths w.r.t. a linear combination as in our own algorithm. To have a state of the art comparison partner for computing the conventional skyline, we compare to ARSC [1]. All tested algorithms start query processing with the same precomputation step to compute query specific lower bounds [16]. To the best of our knowledge this is currently the best published method for performing this task. However, if any better algorithm for lower bound computation is available, all methods could benefit from it.

In a first set of experiment, we compare the number of result paths between conventional and linear path skylines. As already claimed in the introduction, we argue that the set of linearly optimal paths is usually much smaller than the set of all pareto optimal paths. Furthermore, the increase of result paths with an increasing path length and an increasing number of criteria is far less extreme and thus, stays manageable in many applications. Figure 10 displays the size of the path skyline and the linear path skyline in the Munich setting for three and five cost criteria. To measure the distance between the locations, we determine the minimum number of hops between the locations. The plots display the size of the skylines as a function of the number of hops between the locations. It can be seen that the amount of result paths in conventional path skylines increases superlinearly with the number of hops between start and destination. In contrast the size of the linear skyline displays a weak linear increase. Thus, the size of linear path skylines remains manageable even when the query points are far away from each other. We can further observe that the increase w.r.t. the number of criteria, is also far less dramatic as in the case of conventional skylines. While the average amount of routes for five criteria in conventional path skyline increases from 25 for three criteria to 145 for 5 criteria, the linear skyline increases from 5 paths for three criteria to 22 paths for 5 criteria. To conclude, the increase of results in linear path skylines with the distance of the query points and the number of criteria is far less dramatic.

In the next experiment, we compare the runtime of LSCH to ARSC and ExA in the Munich setting for three and five cost criteria (cf. Figure 11). As expected, it can be seen that both dedicated linear path skyline algorithms clearly outperform ARSC when computing the complete path skyline. Thus, both algorithms are able to exploit the fact that they have far less result paths to compute. When comparing ExA to LSCH, the effort needed to compute a linear path skyline is almost identical for three criteria. The reason for the similar behavior is that both algorithms require about the same amount of A*-Searches in the graph and the number of solved linear equations is comparable. For five cost criteria, we clearly see that LSCH performs better than ExA for longer hop distances. ExA has to solve more than 8 times more linear equation systems than LSCH, and the number of linear equation systems in LSCH is still rather small with about 22 updates of the convex hull per query. Additionally, for this setting, ExA starts to encounter queries where the time limit of 360 seconds does not suffice to finish the computation. LSCH on the other hand, does finish all given queries within the time limit.

We additionally test the runtime of all three algorithms on the aforementioned lattice graphs. The results are depicted in Figure 12. It can be seen that in this setting, ExA performs

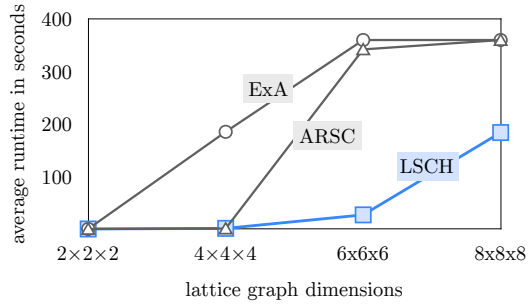
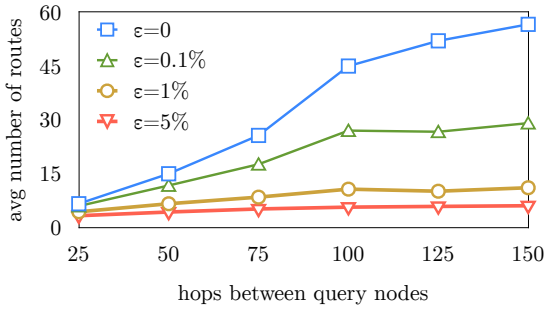
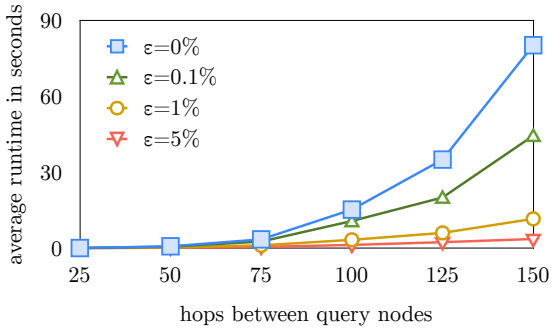


Fig. 12. Runtime comparison of ExA, ARSC and LSCH in the graphs.



(a) Number of Paths



(b) Runtime

Fig. 13. Impact of varying ε-values on the number of result paths and the query runtime

very badly compared to both ARSC and LSCH, even in rather small graphs. The reason for the large query times of ExA is the large number of stages. This results in a dramatic increase of linear equations which have to be solved. LSCH can avoid this increase by using the concept of the convex hull which prunes large amounts of possible search directions.

In a last runtime experiment, we wanted to compare LSCH to an algorithm being dedicated to compute linear path skylines for two criteria. We compared LSCH to BLRSC [2] for the Munich setting. The result can be seen in Figure 14. For smaller distances, both algorithms display a rather similar runtime behavior. Even for larger distances, the advantage of BLRSC is rather small. Thus, the computational overhead of using LSCH compared to BLRSC barely justifies the overhead to implement a specialized solution for the special case of bicriteria networks.

In a final set of experiments, we compare the performance of our approximative ε-linear path skyline algorithms ε-LSCH

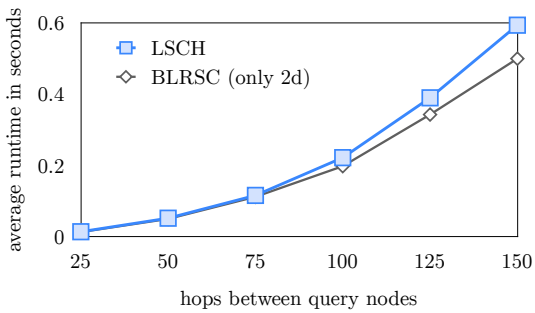


Fig. 14. Runtime comparison of the ordinary LSCH and BLRSC being optimized for bicriterion networks.

with the the exact version. We tested our method with $\varepsilon \in \{5.0\%, 1.0\%, 0.1\%\}$ to show the effect of different ε values. In a first figure, we depict the impact of the ε parameter on the size for the result set. Figure 13(a) shows the decrease in the result paths for all three tolerance levels on the five criteria Munich setting. Even a rather small tolerance of 0.1 percent allows to half the amount the result paths. For the maximal tolerance level of 5%, it can be observed that the number of result paths stays at a very small level with about 6 result paths, even for queries with more than 150 hops between start and target node. Thus, ε-LSCH allows to compute small results that are manageable by users while giving a guarantee on the representativeness of the remaining routes.

After showing that ε-LSCH is suitable to control the amount of result paths by tuning the ε parameter, we now want to investigate whether ε-LSCH can exploit the reduced result set size to speed up query processing. Figure 13(b) illustrates the corresponding runtime in the five criteria Munich setting. As can be seen, ε-LSCH reduces the processing times by the same amount as the number of results. Thus, ε-LSCH allows to compute small representative sets of alternative paths in very efficient time. By tuning the ε parameter, the number of result paths can be reduced in a controlled way. For any given ε there is a guarantee that for any weighting vector there exists a path in the result set which is at most ε percent less cost efficient than the optimal solution.

VI. Conclusions

Path skyline queries compute the set of all pareto optimal solutions between two given locations in a multicriteria network. The cost vector of any pareto optimal is optimal for at least one monotonic combination function mapping the cost vector to a scalar cost value. In this paper, we are interested in the subset of conventional skylines optimal under linear combination functions. We argue that a linear path skyline represents an interesting subset of the corresponding conventional path skyline. Furthermore, the linear path skylines are easier to handle for a user and faster to compute. To directly compute linear path skylines in networks having an arbitrary number of cost criteria, we propose an incremental algorithm for using A*-searches w.r.t. a given linear combination. To prune the search space, we maintain the facets of an augmented convex hull of the cost vectors identified so far. To cope with cases where there has to be a limited amount of representative results, we propose ε-linear path skyline query which computes

a subset of the linear path skyline. The maximum difference between the best cost value in the approximative skyline and the optimal solution of the linear skyline is at most ε. In our experiments, we demonstrate that linear path skylines indeed accomplish the claimed reduction of cardinality even for larger numbers of cost criteria and that our new algorithms LSCH and ε-LSCH show competitive runtimes in street networks and grid graphs. For future work, we plan to investigate cases for other classes of monotonic combination functions. Furthermore, we currently work on extending our method to other types of network queries such as trip planning queries and ranking queries.

REFERENCES

[1] H. P. Kriegel, M. Renz, and M. Schubert, "Route skyline queries: a multi-preference path planning approach," in *Proceedings of the 26th International Conference on Data Engineering (ICDE), Long Beach, CA*, 2010, pp. 261–272.

[2] M. Shekelyan, G. Jossé, M. Schubert, and H.-P. Kriegel, "Linear path skyline computation in bicriteria networks," in *Database Systems for Advanced Applications*. Springer, 2014, pp. 173–187.

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

[4] K. L. Tan, P.-K. Eng, and B. C. Ooi, "Efficient progressive skyline computation," in *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB), Roma, Italy*, 2001.

[5] 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.

[6] 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.

[7] A. J. Skriver, "A classification of bicriterion shortest path (bsp) algorithms," *Asia Pacific Journal of Operational Research*, vol. 17, no. 2, pp. 199–212, 2000.

[8] A. Raith and M. Ehrgott, "A comparison of solution strategies for biobjective shortest path problems," *Computers & Operations Research*, vol. 36, no. 4, pp. 1299–1331, 2009.

[9] Z. Tarapata, "Selected multicriteria shortest path problems: An analysis of complexity, models and adaptation of standard algorithms," *Int. J. Appl. Math. Comput. Sci.*, vol. 17, no. 2, pp. 269–287, Jun. 2007. [Online]. Available: http://dx.doi.org/10.2478/v10006-007-0023-2

[10] M. Ehrgott and X. Gandibleux, "A survey and annotated bibliography of multiobjective combinatorial optimization," *OR-Spektrum*, vol. 22, no. 4, pp. 425–460, 2000.

[11] P. Hansen, "Bicriterion path problems," in *Multiple criteria decision making theory and application*. Springer, 1980, pp. 109–127.

[12] M. Müller-Hannemann and K. Weihe, "Pareto shortest paths is often feasible in practice," in *Algorithm Engineering*. Springer, 2001, pp. 185–197.

[13] B. S. Stewart and C. C. White III, "Multiobjective a*," *Journal of the ACM (JACM)*, vol. 38, no. 4, pp. 775–814, 1991.

[14] B. Yang, C. Guo, C. S. Jensen, M. Kaul, and S. Shang, "Multi-cost optimal route planning under time-varying uncertainty," in *Proceedings of the 30th International Conference on Data Engineering (ICDE),Chicago,IL, USA*, 2014.

[15] E. Machuca and L. Mandow, "Multiobjective heuristic search in road maps," *Expert Syst. Appl.*, vol. 39, no. 7, pp. 6435–6445, Jun. 2012. [Online]. Available: http://dx.doi.org/10.1016/j.eswa.2011.12.022

[16] C. Tung Tung and K. Lin Chew, "A multicriteria pareto-optimal path algorithm," *European Journal of Operational Research*, vol. 62, no. 2, pp. 203–209, 1992.

[17] A. Raith and M. Ehrgott, "A comparison of solution strategies for biobjective shortest path problems," *Computers & Operations Research*, vol. 36, no. 4, pp. 1299–1331, 2009.

[18] J. Mote, I. Murthy, and D. L. Olson, "A parametric approach to solving bicriterion shortest path problems," *European Journal of Operational Research*, vol. 53, no. 1, pp. 81–92, 1991.

[19] J. Brumbaugh-Smith and D. Shier, "An empirical investigation of some bicriterion shortest path algorithms," *European Journal of Operational Research*, vol. 43, no. 2, pp. 216–224, 1989.

[20] A. J. Skriver and K. A. Andersen, "A label correcting approach for solving bicriterion shortest-path problems," *Computers & Operations Research*, vol. 27, no. 6, pp. 507–524, 2000.

[21] E. Machuca and L. Mandow, "Multiobjective heuristic search in road maps," *Expert Systems with Applications*, vol. 39, no. 7, pp. 6435–6445, 2012.

[22] Ö. Özpeynirci and M. Köksalan, "An exact algorithm for finding extreme supported nondominated points of multiobjective mixed integer programs," *Management Science*, vol. 56, no. 12, pp. 2302–2315, 2010.

[23] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*. Springer, 1985.

[24] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, "The quickhull algorithm for convex hulls," *ACM Transactions on Mathematical Software (TOMS)*, vol. 22, no. 4, pp. 469–483, 1996.

[25] Ö. Özpeynirci and M. Köksalan, "An exact algorithm for finding extreme supported nondominated points of multiobjective mixed integer programs," *Management Science*, vol. 56, no. 12, pp. 2302–2315, 2010.

[26] P. McMullen, "The maximum numbers of faces of a convex polytope," *Mathematika*, vol. 17, no. 02, pp. 179–184, 1970.

[27] R. Seidel, "The upper bound theorem for polytopes: an easy proof of its asymptotic version," *Computational Geometry*, vol. 5, no. 2, pp. 115–116, 1995.