Preference-bound Skyline Routing

Thesis · November 2022

DOI: 10.13140/RG.2.2.33443.32806

1 author:

Patrik Thomas Michalski

Christian-Albrechts-Universität zu Kiel

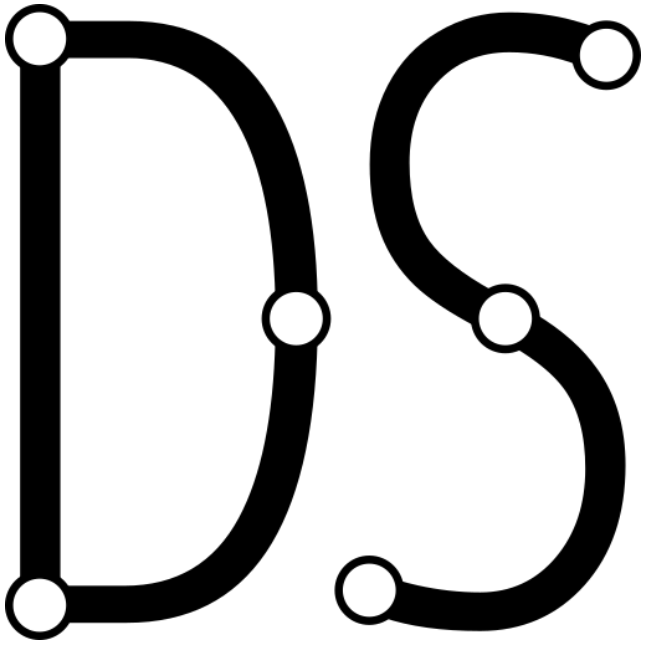
4 PUBLICATIONS **0** CITATIONS

SEE PROFILE

Master Thesis

Preference-bound Skyline Routing

Michalski, Patrik Thomas



Technical Faculty
Department of Computer Science
Archaeoinformatics - Data Science Research Group
University of Kiel
Kiel, Germany 2022

Preference-bound Skyline Routing
Michalski, Patrik Thomas

Advised by:
First supervisor: Prof. Dr. Renz, Matthias (Dr. rer. nat., habil)
Second supervisor: M.Sc. Preuß (Amann), Niko

Master Thesis
Technical Faculty
Department of Computer Science
Archaeoinformatics - Data Science Research Group
University of Kiel
Kiel, Germany
Telephone: +49 431 880-7270

Eidesstattliche Erklärung

Hiermit erklärt ich - Michalski, Patrik Thomas - an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Hilfsmittel und Quellen verwendet habe. Weiterhin versichere ich, dass die eingereichte schriftliche Fassung der digitalen Fassung entspricht.

Kiel, den 28. November 2022

"The greatest enemy of knowledge is not ignorance,
it is the illusion of knowledge."

written by Hawking, Stephen.

Acknowledgments

I would like to thank my family, who have always supported me, especially my parents, Katharina and Waldemar Michalski, for their endless love and support. Without you both, I would not be the person I'm today! - I love you!

Further, I want to thank the colleagues I met and friends I have made, who have helped me with their guidance and have inspired me over the past years.

Additionally, a big thank you goes to my advisors Prof. Dr. Renz, Matthias, and M.Sc. Preuß (Amann), Niko, for being an endless source of knowledge! Thank you for your help whenever I encountered misconceptions or problems. Furthermore, I am thankful for our working group's warm-hearted atmosphere!

Conclusory, thank all those who have helped to improve this thesis. Without you, there would be more mistakes and misunderstandings. You are marvelous!

Michalski, Patrik Thomas; Kiel, November 2022.

Preference-bound Skyline Routing
Michalski, Patrik Thomas
Department of Computer Science
University of Kiel

Abstract

In today's globalized industry, efficient distribution of goods is essential for fighting against global warming. Furthermore, efficient planning saves money in an environment with increasing energy costs and scarcer getting resources. Since shipping transports about 80 % of goods, ocean shipping is the backbone of global trade and an integral part of the supply chain for most industries while contributing to climate change and global warming.
Recently, this problem has become more relevant in economy and politics, e.g., by developing new methods to reduce the influence of greenhouse gases and new laws limiting the amount of released CO_2 through carbon taxes.

Since other methods, e.g., shortest-path algorithms, do not allow incorporating unknown external constraints, e.g., safety paths or statutory provisions, we use skyline queries for routing to find the best-fitting Pareto-optimal shipping routes. We utilize Linear Route Skyline Queries (LRS-Qs) while considering environmental conditions such as currents, waves, wind, and a given preference, allowing us to leverage the Advanced Route Skyline Computation (ARSC) algorithm to use Linear Preference Boundaries (LPBs). Thus, the user can select from various routes best fitting their preference, e.g., the most climate-friendly route. Thereby, we can show that our approach improves the overall runtime in different real-world scenarios while improving the importance of the result since only those routes remain that best fit the user preference. Further, Linear Preference Boundaries (LPBs) tremendously shrink the possible search space in large multi-edged graphs since we can prune candidates faster.

Keywords: multi-attribute and -objective optimization and routing in graphs, Advanced Route Skyline Computation (ARSC), Linear Route Skyline (LRS), and Linear Preference Boundary (LPB).

Contents

List of Algorithms

List of Figures

List of Tables

Chapter 1
Introduction

In today's globalized industry, efficient distribution of goods is essential for fighting against global warming. Furthermore, efficient planning saves money in an environment with increasing energy costs and scarcer getting resources. Since shipping transports an overwhelming majority of goods, ocean shipping is the backbone of global trade and an integral part of the supply chain for most industries while contributing to climate change and global warming [1]. During the last 30 years, the amount of cargo transported almost tripled, from four to nearly 10.7 billion tons [1], as shown in Figure 1.1. At the same time, global merchant fleet increased significantly by almost two million deadweight tons. However, the flourishing development of the shipping industry comes with various impacts, particularly the global gas emissions from shipping increased by about 9.6 % from 2012 to 2018 [1]. Thereby, most emissions of ships are carbon dioxide and methane, which accelerate climate change.
Recently, this problem has become more relevant in economy and politics, e.g., by developing new methods to reduce the influence of greenhouse gases and new laws limiting the amount of released CO_2 through carbon taxes.

This thesis provides a method to find the best-fitting Pareto-optimal shipping routes, leveraging the Advanced Route Skyline Computation (ARSC) algorithm to utilize Linear Preference Boundaries (LPBs) while considering environmental conditions, such as currents, waves, and wind, providing a real-world approach. Thus, the user can select from various routes best fitting their preference, e.g., the most climate-friendly route.
Since other methods, e.g., shortest-path algorithms, do not allow incorporating unknown external constraints, e.g., safety paths or statutory provisions, we use Linear Route Skyline Queries (LRS-Qs) for routing to find the best-fitting Pareto-optimal shipping routes.

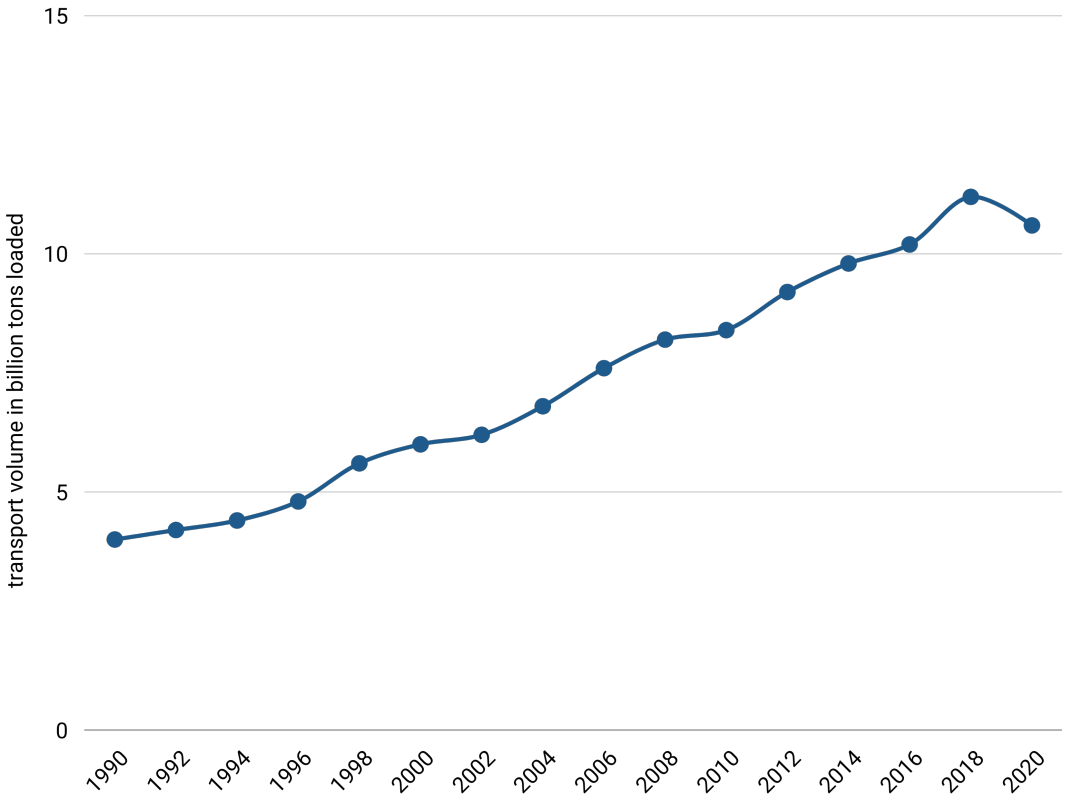


Figure 1.1: Visualization of the global seaborne trade from 1990 to 2020 [1].

The research community introduced various methods for processing skyline queries in multi-dimensional databases. It became a critical operation when searching for ranked result objects. Instead of considering a fixed weighting for a set of optimization criteria, the skyline method retrieves all objects being optimal w.r.t. any arbitrary linear weighting of the underlying criteria.
For example, consider a traveler seeking a route between two cities with low fuel consumption and short driving time among the available routes. The skyline method would retrieve those routes that are Pareto-optimal. However, since the traveler might need help determining a fixed ratio between both characteristics, the best choices should consider all possible ratios. Hence, the quality of all possible cost combinations is the weighted sum of all values over all considered criteria. Thus, the higher the weight, the higher the importance of the corresponding attribute, which we call linear preference.
In many applications, storing data as a graph is the only solution for efficiently accessing information. Thereby, computing cost-optimal paths between two nodes are essential for distance computation and routing. For example, in public transportation and road networks, computing cost-optimal routes are the main functionality of navigation.

In other networks, such as small-world networks, the distance between nodes measures the closeness between people. A shorter distance means that both people are more related, while a more significant distance indicates that they may not know each other. Depending on the given application, the cost of traversing an edge has a different meaning. While skyline queries compute the set of Pareto-optimal paths between two given nodes, the number of skyline paths increases exponentially with the distance between the nodes and the number of cost criteria [2]. Furthermore, the result set is often too big to be used because confronting a user with many alternatives is often not very helpful. Thus, reducing the set of results to those helpful makes sense.

We will extend the Advanced Route Skyline Computation (ARSC) algorithm with a multi-attribute route-skyline query utilizing a preference-bound pruning strategy for faster pruning of routes while leading to more valuable results.

In this thesis, we present the following contributions:

- We introduce the concept of preference-bound pruning in the context of route-skyline queries. We propose a new approach for determining Linear Route Skylines (LRSs) in dynamic and uncertain search spaces.
- Further, we discuss Linear Preference Boundaries (LPBs) as a pruning filter for Linear Route Skylines (LRSs).
- Finally, we conclude with an analysis of our extension.

We organized the remains of the thesis as follows. First, Chapter 2 classifies our proposed work and shows different approaches for solving associated problems while presenting recent studies on the Linear Route Skyline (LRS) concept. Then, in Chapter 3, we cover all relevant concepts, including all background knowledge and the terms and definitions needed to understand our approach. Following, in the Chapter 4, we present our primary contribution, introducing the concept of Linear Preference Boundaries (LPBs) as well as our proposed algorithm. Moreover, we talk about the implementation of our approach, the requirements we have chosen, and the problems that arise while implementing Linear Preference Boundary Skyline Queries (LPBS-Qs). Chapter 5 discusses how Linear Preference Boundary Skyline Queries (LPBS-Qs) impact the overall runtime and what impact the length and position of an α-Range have on the behavior of our approach. Further, we show if it is worth spending more time to find a suitable α-Range as a preprocessing step to improve finding more fitting Linear Route Skylines (LRSs). Concluding with Chapter 6, we give a conclusion about our primary contribution and present a selection of possible directions for further improvements and research in this field.

Chapter 2
Related Work

This chapter discusses current state-of-the-art approaches. These approaches consider applying skyline queries in databases, i.e., complete and incomplete, or road networks. However, we utilize skyline queries in a different context since we use them on ocean networks (see Section 2.1). Therefore we will cope with several problems that do not appear in the below-mentioned methods, as discussed in Section 4.2.1.
Furthermore, we mention different strategies for applying linearity to improve pruning during exploration (see Section 2.2). Finally, we close this chapter with an overall comparison of our approach while leading to our main contribution and implementation chapters.

2.1 The Skyline-operator

Borzsonyi et al. [3] introduced skyline queries while proposing several variants, i.e., block-nested-loop processing and a divide-and-conquer approach. However, since skyline processing has attracted considerable attention, multiple approaches to computing skylines have followed. Finding a skyline is an NP-hard problem since the length of a path in a skyline may increase exponentially with the number of hops between the source and destination node. Nevertheless, Müller-Hannemann and Weihe [4] showed that the number of paths is feasibly low when using strongly correlated cost criteria.
To overcome drawbacks like results are not personalizable, reporting the same result, and the overwhelming output size, Mouratidis et al. [5] introduced a method to combine skyline and top-k queries to allow practical decision support, where $k \in \mathbb{N}$. This approach allows controllable output size, flexibility in preference specification, and personalization. Thereby, they introduced two operators, ORD and ORU. ORD employs an adaptive notion of dominance, while ORU sticks closer to a ranking.

We start with the approaches of Chomicki et al. [6], Morse et al. [7], and Tan et al. [8]. They proposed similar ideas of progressive methods to improve the approach introduced by Borzsonyi et al. [3]. Tan et al. [8] proposed an index method, which divides the dataset into d-dimensional sorted lists with d optimization criteria, where $d \in \mathbb{N}$. Another method utilizes bitmaps to describe those datasets with lower cardinality domains, while describing each optimization criterion, they use a small set of discrete attribute values.
A different approach introduced by Kossmann et al. [9] processes recursively a Nearest-Neighbor Query (NN-Q) using an R-Tree. They start by finding the nearest neighbor of the query point, which has to be part of the skyline and using it to prune those paths that belong to the query point's square. Afterward, the same process repeats on the remaining data recursively. A problem that can arise is that these remaining sections might overlap, leading to inconsistency. Therefore, Papadias et al. [10] proposed this improvement. In addition, they proposed a branch and bound approach, which is guaranteed to visit each page of the underlying R-Tree at most once.
Köhler et al.'s [11] approach divide the graph into regions while gathering information on whether an edge is on the shortest path leading to a specific region.
Other strategies use post-processing methods for selecting the resulting skyline paths. For example, Chan et al. [12] proposed a k-dominated skyline query, where $k \in \mathbb{N}$. Furthermore, they generalized the dominance relationship by requiring a path to improve all other paths in at least k attributes.
A different approach tries efficiently to parallelize the computation of skylines for subsets of all optimization criteria [13], [14].

Usually, routing for road networks is based on finding the shortest path between two nodes using one of the best-known algorithms, Dijkstra's [15] shortest-path algorithm. In contrast, applying heuristics with the A$\star$ (A-star) search algorithm allows to prune those routes faster and reduces the search space while improving the graph expansion.

Unfortunately, all these approaches need to be more comprehensive regarding multi-preference-bound routing. Furthermore, we use in our setting an understanding of route-skylines where we are interested in finding those routes whose cost vectors are non-dominated by any other route between the same two nodes in a Multi-attribute Network Graph (MAG). Therefore, we will extend the following approach by Kriegel et al. [16] with a multi-attribute route-skyline query utilizing a preference-bound pruning strategy for faster pruning of routes while leading to more valuable results.

Kriegel et al. [16] introduced one of the state-of-the-art labeling correcting algorithms using lower bounds for computing route-skylines. Thereby, they prune those routes that are not further extendable into a result skyline and whose intermediate nodes are dominated by another route ending in the same destination node. They employ a specific Lipschitz embedding for lower-bound approximation utilizing reference nodes while using the approximated remaining costs to the destination node to prune those routes already worse than any currently known route from the set of result skylines. Stewart and White [17] and Yang et al. [18] introduce similar methods.
A similar approach by Machuca and Mandow [19] shows that computing lower-bound costs individually for each query is also possible.

2.2 Pruning with linearity

In this section, we survey related problems to Linear Route Skylines (LRSs) as an extension to the previously reviewed approaches, constituting a superset of Linear Route Skyline Queries (LRS-Qs).

We start with the approach of Özpeynirci and Köksalan's [20] to find non-dominated points of multi-attribute mixed integer programs. Their algorithm searches all subsets of cost vectors with a given cardinality and establishes a hyper-plane in the solution space. When processing such a subset, a linear equation system determines the plane's normal vector. Afterward, this normal vector determines whether it is necessary to compute a new path. A drawback of this procedure is that the number of subsets can dramatically increase while increasing the number of linear equations that need to be solved [2].
To cope with this problem, Shekelyan et al. [2] introduced their concept of linear skylines using facets on a convex hull instead of subsets. Although computing facets causes an additional step in the algorithm, there are fewer facets than subsets. Thus, the number of linear equation systems to solve is considerably smaller [2].
The second approach of Shekelyan et al. [21] and Mote et al. [22], introduced as a two-phase method of determining all non-dominated paths, computes in the first phase the supported solutions. Thereby, phase two determines the linear skyline. However, the most significant drawback is that they heavily rely on the characteristics of a two-dimensional cost space [23], [24].

Chapter 3
Terms and Definitions

This chapter covers all relevant concepts, including all background knowledge and the terms and definitions needed to understand our approach.

Overall, this chapter describes the following:

- Pareto-optimum and -domination (see Section 3.1)
- The Skyline-operator (see Section 3.2)
- Multi-attribute Network Graph (MAG) (see Section 3.3)
- Advanced Route Skyline Computation (ARSC) (see Section 3.4)

3.1 Pareto-optimum and -domination

Pareto-optimality is a core concept in the optimization field and measures the efficiency in the multi-objective context. A Pareto-optimal solution is unique in single objective optimization problems, focusing on the decision variable space. In the context of the multi-objective optimization process, it extends the theory by allowing single objectives to be optimized simultaneously to account for several conflicting objectives. It is a mathematical process of looking for alternatives representing the Pareto-optimal objects. Thereby, the set of non-dominated objects in the objective space defines a boundary beyond which no further improvements are feasible without worsening at least one of the other objectives. Formally speaking, the definition of dominance between two objects is shown in Definition 1.

Definition 1 (Pareto-optimum and -domination)**.** Let Ω be a set of objects with dimensionality d, where ω, $\omega' \in \Omega$ and $d \in \mathbb{N}$. Then ω dominates ω' iff:

$$\forall\ 0 \leq \mathrm{i} \leq d \text{ - } 1\text{: } \omega_{\mathrm{i}} \leq \omega'_{\mathrm{i}} \wedge \exists\ 1 \leq \mathrm{j} \leq d \text{ - } 1 \text{ with } \mathrm{i} \neq \mathrm{j}\text{: } \omega_{\mathrm{j}} < \omega'_{\mathrm{j}}.$$

In other words, an object is Pareto-optimal if no other object improves the value of any objective criteria without worsening at least one other criterion. Therefore, an object is said to be non-dominated or Pareto-optimal if no other object dominates.
Furthermore, the set of all non-dominated objects forms a Pareto-front, representing the optimal trade-off between all objectives as shown in Figure 3.1.

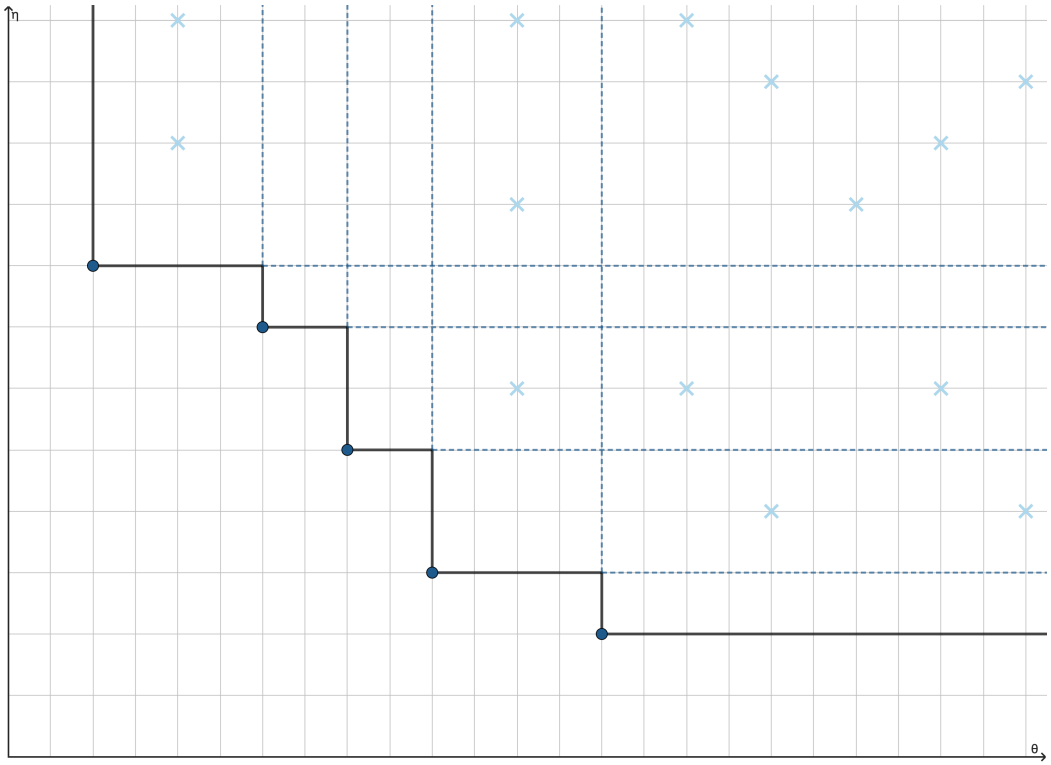


Figure 3.1: Visualization of a Pareto-front (black line) in a Euclidean space, representing the optimal trade-off between all objectives. The dark blue points dominate all light blue crosses in at least one dimension, while the blue dashed lines visualize the space that is pruned by each non-dominated point.

Let us consider the following example: For a system with n parameters, where $n \in \mathbb{N}$, and their particular values of the parameters σ_0, σ_1, …, σ_{n-1} result in particular values of criterion functions λ_0, λ_1, …, λ_{n-1}, which are functions of the inputs and measure the performance of the system. Each selection of parameters yields a feasible solution in the objective space, the complete set of allowable solutions. In general, most of these feasible solutions will be dominated, where it is better in at least one objective and at least as good in all other objectives. Only few solutions will be non-dominated where the value of a given objective function is only further improvable at the cost of at least one other objective. These leftovers are Pareto-optimal and create a Pareto-front for the introduced system, as exemplary shown in Figure 3.1.

3.2 The Skyline-operator

Skyline queries are essential for several applications involving multi-attribute decision-making [2]. This method finds those Pareto-optimal objects that most fit the user's preference, lying on the Pareto-front (see Section 3.1).

Mathematically, Definition 2 gives the definition of a skyline.

Definition 2 (The Skyline-operator)**.** Let Ω be a set of objects. Then:

$$\{\omega \in \Omega \mid \nexists\, \omega' \in \Omega\text{: } \omega' \text{ dominates } \omega\}$$

is referred to as skyline of Ω.

Skylines are related to several well-known problems, e.g., convex hulls, ranked or top-k queries, and Nearest-Neighbor Query (NN-Q) searches:

- The convex hull contains the subset of skyline points that may be optimal only for linear preference functions as opposed to any monotone function.
- Ranked or Top-k queries retrieve the best k objects, where $k \in \mathbb{N}$, that minimize a specific preference function. They differ from skyline queries insofar that the output changes according to the input function, and the retrieved points are not guaranteed to be part of the skyline.
- NN-Q specify a query point ω and output the objects closest to ω' in increasing order of their distance, where ω, $\omega' \in \Omega$. Thereby, those objects have to be indexed by some data-partition methods. Entries, which are farther than the nearest neighbor, are pruned. As a result, the first nearest neighbor searches show that the first nearest neighbor is always part of the skyline.

3.3 Multi-attribute Network Graph

This section explains the idea of Multi-attribute Network Graphs (MAGs) while discussing all terms and definitions (see Sections 3.3.1 and 3.3.2) needed to understand them in our settings.

3.3.1 An Introduction to MAGs

A Multi-attribute Network Graph (MAG) is a directed graph, as shown in Definition 3, but instead of having only just one weight between two adjacent nodes, a weight vector represents the weight for d dimension, where $d \in \mathbb{N}$. Thereby, each of the d-dimensional weight vectors contain d attributes that determine the costs for each dimension within the graph.

A typical use case for a MAG is a road network. Thereby, the nodes could correspond to crossings, the edges correspond to roads, and the weight vectors describe the considered attributes of each road. The attributes of a road might represent, e.g., the admitted speed or the time needed to cross the road.

Definition 3 (Multi-attribute Network Graph (MAG))**.** A multi-attribute graph with d-dimensions is a directed graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$, where $\mathbb{V}$ denotes the set of vertices, $\mathbb{E} \in \mathbb{V} \times \mathbb{V}$ denotes the set of directed edges, and $d \in \mathbb{N}$.

Thereby, the following holds:

- The dimensionality refers to the number of cost functions while in a graph with d-dimensions, d cost functions $\lambda_0, \ldots, \lambda_{d-1}$: $\mathbb{E} \mapsto \mathbb{R}_{\geq 0}$ exits.
- Further, a d-dimensional weight vector of an edge $(u, v) \in \mathbb{E}$ is defined by $\lambda((u, v)) \coloneqq [\lambda_0(u, v), \ldots, \lambda_{d-1}(u, v)]^\intercal$, where $u, v \in \mathbb{V}$.
- A path $\rho \in \mathbb{P}$ is a consecutive set of edges which does not visit any node twice, where $\mathbb{P}$ is the set of all paths in $\mathbb{G}$. Furthermore, a path has a d-dimensional weight vector $w = (w_0, \ldots, w_{d-1}) = \Sigma_{(u, v) \in \rho}$: $\lambda((u, v))$, which is the component-wise sum of its edges.

3.3.2 Multi-attribute Pareto-optimum and -domination

As mentioned in Section 3.1 and shown in Definition 1 an object dominates another object if this object improves the value of one objective criterion without worsening all other criteria. Thereby, the exact definition is also valid in the multi-attribute case.
Definition 4 shows the extended Definition 1 introduced in Section 3.1 with incorporated multi-attributes.

Definition 4 (Multi-attribute Pareto-optimum and -domination)**.** Let Ω be a set of objects with dimensionality d, where $\omega, \omega' \in \Omega$ and $d \in \mathbb{N}$. Furthermore, let d' be the selection of the d'-th dimension of the weight vector of ω and ω', where $d' \in \mathbb{N}$. Then ω dominates ω' in the d'-th dimension iff:

$$\exists\ 0 \leq d' \leq d - 1\text{: } \omega_{d'} \text{ dominates } \omega'_{d'}.$$

An object ω dominates object ω' if it yields at least an improvement w.r.t. one attribute and ω is at least as good as ω' for all other attributes.

3.4 Advanced Route Skyline Computation

In this section we introduce the algorithm we used as the foundation for our extension while covering all the crucial parts.

3.4.1 An Introduction to the ARSC algorithm

Recently, studies introduced methods for processing route-skyline queries in multi-dimensional databases, i.e., complete and incomplete. As previously mentioned, the skyline method retrieves all objects being optimal w.r.t. an arbitrary linear weighting of the underlying criteria (see Section 3.2). However, considering different preferences involves different routes, a skyline-fashioned answer with relevant route candidates is highly useful.
Kriegel et al. [16] proposed a new highly dynamic approach for computing skylines on routes in a road network graph with previously unknown source and destination nodes, considering multiple preferences, e.g., fuel consumption and driving time, by significantly reducing search space and efficiently computing the skyline, called Advanced Route Skyline Computation (ARSC).
Moreover, they employed a particular Lipschitz embedding technique to enable a best-first-based graph exploration considering route preferences based on any arbitrary road attributes (see Section 3.4.2). Thereby, they iteratively compute the top routes according to at least one given preference efficiently, avoiding multiple computations of that route from scratch in each iteration. Thus, their algorithm dynamically explores and prunes those routes that will not lead into a skyline. Further, they based their algorithm on a lower-bound forward estimation for each optimization criterion to efficiently compute route-skylines. Therefore, if another route to the destination already dominates this optimistic forward estimation, the algorithm stops further extending this route.
Furthermore, they proposed two pruning techniques to reduce the search space. Their pruning strategies aim to prune as many route candidates as possible during the graph exploration. To achieve this, they prune candidates during exploration and remove those routes which do not meet the requirements.
Kriegel et al.'s [16] approach handle several problems processing route-skyline queries on multi-attribute databases compared to established solutions. First, the set of all possible routes between two nodes cannot be assumed to be previously known. Instead, the MAG stores all information implicitly, so the routes must be derived before determining their costs. A naive solution could be calculating all routes and sorting out all dominated ones. However, the number of possible routes increases exponentially with network distance [2]. Another problem with this approach is that the number of possible sources and destination nodes can be rather large.

Moreover, in databases where there is only one skyline for one database, a MAG allows a large number of different skylines corresponding to all possible source and destination nodes. As a result, precomputing the set of all routes would cause an enormous amount of data and, thus, make it infeasible for graphs exceeding a specific size. A different idea would be to organize the objects in efficient data structures, e.g., R-Tree, but again the problem of materialization arises. Thus, building an index structure is only possible by knowing the objects.
Finally, due to the large number of routes connecting two nodes in a MAG, an efficient solution has to prevent the calculation of dominated routes as early as possible. Therefore, pruning has to happen as early as possible. As a result, the existing methods for route-skyline computation with multi-attributes do not apply.

3.4.2 Lipschitz Embedding

The ARSC algorithm needs two graphs to find efficient route-skyline routes: a Lipschitz embedding and an underlying network graph.
Dijkstra [15] proposed one of the most well-known approaches for shortest-path computation. This algorithm finds the optimal solution when assuming that no additional information about the shortest path is available, with the drawback that it usually has to consider large portions of the graph. However, to overcome this drawback and to further speed up shortest-path computation, Kriegel et al. [16] utilize an optimistic approximation for each point in the network for the remaining distance to the destination. By applying this lower-bound approximation in combination with the already known costs of the currently explored path, it is possible to check if it is still possible to reach the destination via an extension of this path shorter than the current shortest path. Therefore, it is possible to prune a path if there is an already known path to the destination which is already shorter than the currently examined path. At the same time, it significantly reduces the number of traversed nodes for shortest-path computation. Thereby, a simple solution for calculating a lower-bound approximation is to employ Euclidean distance. Thus, the shortest path would always be the direct line, so traversing on the road network would be consistently equal or higher to this distance.
Unfortunately, this lower bound is only extendable to criteria correlated to the spatial distance. Thus, applying a search with the A$\star$ (A-star) search algorithm on an arbitrary preference function combining various optimization criteria needs a more general approach. They employed a Lipschitz embedding of the MAG using a singleton reference set called reference nodes.

Thereby, the Lipschitz embedding transforms the nodes of a given MAG into $m \times n$-dimensional cost vectors, where m denotes the dimensionality of the cost vectors of the graph, n is the number of reference nodes, and m, $n \in \mathbb{N}$. Constructing a Lipschitz embedding for a MAG includes calculating all shortest paths between each node with their attributes and all reference nodes. Since the graph structure remains static, the Lipschitz embedding is processed offline by keeping all results within each node.

After the creation of the Lipschitz embedding as a preprocessing step, the ARSC algorithm uses this information to determine a lower-bound estimation. Thereby, they can utilize the lower-bound estimation to prune routes already during the route exploration. The essential idea behind this strategy is to apply the forward cost estimator of a search with the A$\star$ (A-star) search algorithm for a partially explored sub-route ρ to estimate the cost vector of a thoroughly explored route ρ' that contains the sub-route ρ, where ρ, $\rho' \in \Upsilon$ and Υ is the set of routes. Furthermore, they extend pruning by letting all nodes prune those sub-routes dominated by another sub-route ending within themself while achieving an efficient approach for determining Pareto-optimal solutions.

Chapter 4
Preference-bound Routing

In this chapter, we discuss the theoretical insights of our contribution stated in the Introduction 1. First, we introduce the concept of linear skylines while extending it to the concept of Linear Route Skylines (LRSs) (see Section 4.1.1) and Linear Preference Boundaries (LPBs) (see Section 4.1.3) while justifying our assumptions, including examples and their pseudocode. Next, we discuss our extension of Linear Preference Boundary Skyline Queries (LPBS-Qs) and provide the pseudocode (see Section 4.1.5). We conclude this chapter with a discussion about the implementation and the problems that arose during the concept phase (see Sections 4.2.1 and 4.2.2), noting our Skyline-routing framework, which contains our implementation (see Section 4.2.3).

4.1 Theoretical insights

In the following sections, we provide and discuss the theoretical insights into Linear Preference Boundaries (LPBs). We start with the concept of linearity in the context of skylines.

4.1.1 Linear Skylines and Linear Route Skylines

In order to distinguish between skylines introduced in the Section 3.2 and linear skylines, we define the concept of linear skylines.

Definition 5 (Linear Skyline)**.** Let Ω be a set of objects. Then:

$$\forall\ \mu \in \mathbb{R}_{>0}\colon\ \exists\ \omega \in \Omega\colon\ \forall\ \omega' \in \Omega\colon\ \mu^\intercal \times \omega = \min(\mu^\intercal \times \omega')\ \text{(completeness)},$$
$$\forall\ \omega, \omega' \in \Omega \text{ with } \omega \neq \omega'\colon\ \exists\ \mu \in \mathbb{R}_{>0}\colon\ \mu^\intercal \times \omega < \mu^\intercal \times \omega'\ \text{(minimality)}$$

the set containing those ω objects is referred to as linear skyline of Ω.

Definition 5 is similarly extendable for multi-attributes as Definition 4.

It follows directly from the definition that the set of LRSs is a subset of the conventional skyline since a linear skyline contains only those objects that are optimal under a linear combination. Figures 4.1 and 4.2 visualize this property.

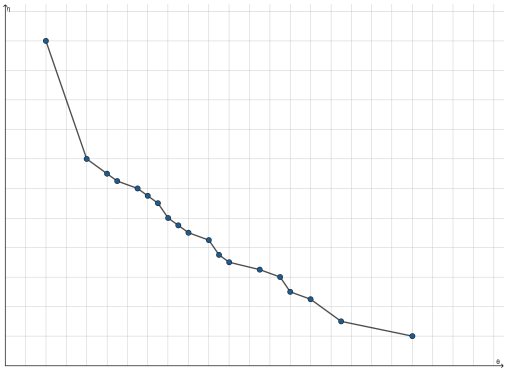


Figure 4.1: Conventional skyline which contains only those objects that are optimal under any monotonic cost function.

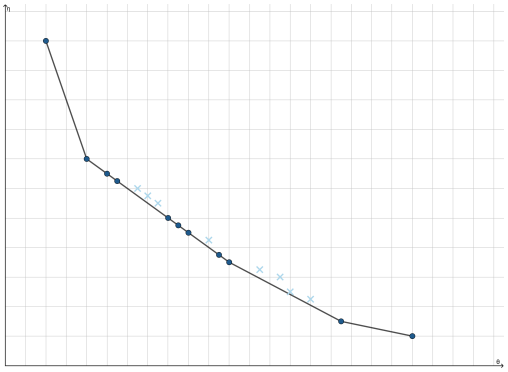


Figure 4.2: Corresponding linear skyline which contains those objects that are optimal under any linear combination.

4.1.2 Computing LRSs

As introduced by Shekelyan et al. [2], building a convex hull and computing facets while using normal vectors retrieves linear skylines. They incrementally check if an object is part of the linear skyline while determining facets of the convex hull and their normal vectors. An object is part of the linear skyline if it is smaller than the minimal dot product with the facet's vertices. They proceed as long as the set of open facets is non-empty. If the set of open facets is empty, the hyperplanes of the closed facets define the convex hull, and the algorithm terminates. The result is a complete linear skyline.

The main drawback of this proposed approach is that the search space needs to be already known, which is impossible in our setting since this constraint follows directly out of the mode of operation of how the ARSC algorithm works. The ARSC algorithm is an exploration algorithm that tries to minimize the lookup and calculation of unwanted routes. Moreover, knowing all costs would make a global calculation of all shortest paths from the source node to all other nodes of the MAG necessary. Since it would directly make sense to use skyline queries redundant further, as a consequence, the ARSC algorithm would be useless and replaced by Dijkstra's [15] shortest-path algorithm. Further, the runtime for large MAGs would not be practicable for real-world usage since routing should be adaptable and fast, especially for ship routing, where the condition can change fast. We need, therefore, a different approach.

Our approach determines an LRS similar to Shekelyan et al. [2] using hyperplanes and normal vectors to determine if a route is part of the LRS. However, instead of using facets, we use the signed shortest distance from the nearest neighbors of the route's cost vector to the cost vector we are checking. Thus, we compute the signed shortest distance to check which side of the hyperplane our route's cost vector belongs, as shown in Definition 6.

Definition 6 (Linear Route Skyline (LRS))**.** Let Υ be a set of routes with a 2-dimensional cost vector, ρ, ρ', and $\rho'' \in \Upsilon$, and let $\lambda(\cdot)$: $\Upsilon \mapsto \mathbb{R}_{\geq 0}$ be a 2-dimensional cost function. ρ is part of the Linear Route Skyline (LRS) iff:

$$(\lambda_0(\rho) \text{ - } \lambda_0(\rho')) \cdot (\lambda_1(\rho'') \text{ - } \lambda_1(\rho')) \text{ - } (\lambda_1(\rho) \text{ - } \lambda_1(\rho')) \cdot (\lambda_0(\rho'') \text{ - } \lambda_0(\rho')) > 0.0.$$

We provide the mathematical reasoning in the following:

Remark 1 (Linear Route Skyline (LRS)). Let Υ, ρ, ρ', and $\rho'' \in \Upsilon$ be given, as in Definition 6. Further, let $\lambda(\cdot)$: $\Upsilon \mapsto \mathbb{R}_{\geq 0}$ be a 2-dimensional cost function. The direction of the line $\langle \rho' \rho'' \rangle$ is defined by $\langle \lambda_0(\rho'') \text{ - } \lambda_0(\rho'), \lambda_1(\rho'') \text{ - } \lambda_1(\rho') \rangle$. Further, the perpendicular direction to this line is given by $\overrightarrow{\mathrm{n}}$ and defined by $\langle \lambda_1(\rho'') \text{ - } \lambda_1(\rho'), \text{-}(\lambda_0(\rho'') \text{ - } \lambda_0(\rho')) \rangle$. One possible vector going from $\langle \rho' \rho'' \rangle$ to ρ is $\overrightarrow{\mathrm{m}}$ and given by $\lambda(\rho) \text{ - } \lambda(\rho') = \langle \lambda_0(\rho) \text{ - } \lambda_0(\rho'), \lambda_1(\rho) \text{ - } \lambda_1(\rho') \rangle$. This vector is made out of two components, a component $\overrightarrow{\mathrm{m}}_{\parallel}$ that is parallel and a component $\overrightarrow{\mathrm{m}}_{\perp}$ that is perpendicular to the line $\langle \rho' \rho'' \rangle$.
We are essentially interested in knowing if $\overrightarrow{\mathrm{m}}_{\perp}$ has the same direction as $\overrightarrow{\mathrm{n}}$. Therefore, we have to determine the sign of the dot product of $\overrightarrow{\mathrm{m}}$ and $\overrightarrow{\mathrm{n}}$ since $\overrightarrow{\mathrm{m}} \cdot \overrightarrow{\mathrm{n}} = (\overrightarrow{\mathrm{m}}_{\parallel} + \overrightarrow{\mathrm{m}}_{\perp}) \cdot \overrightarrow{\mathrm{n}} = \overrightarrow{\mathrm{m}}_{\perp} \cdot \overrightarrow{\mathrm{n}}$. By definition $\overrightarrow{\mathrm{m}}_{\parallel} \cdot \overrightarrow{\mathrm{n}}$ is 0 because a direction parallel to a line will be perpendicular to the normal of that line. Applying the dot product yields Definition 6.

Let us consider the following example: As shown in Figure 4.3, we already have a partial LRS and want to know if ρ is part of the LRS, where $\rho \in \Upsilon$. First, we build a hyperplane using the nearest neighbors, as shown in Figure 4.4. Next, we use Definition 6 to check if the object points in the same direction as the normal vector $\overrightarrow{\mathrm{n}}$. Figure 4.5 shows that the calculation yields that the new object belongs to the LRS. We can add it to the LRS. Finally, Figure 4.6 shows how the calculation yields a non-linear object that does not belong to the LRS.

We conclude this section by providing the Pseudocode 4.1 for determining an LRS. However, to further extend pruning, we introduce in the following section the concept of Linear Preference Boundaries (LPBs).

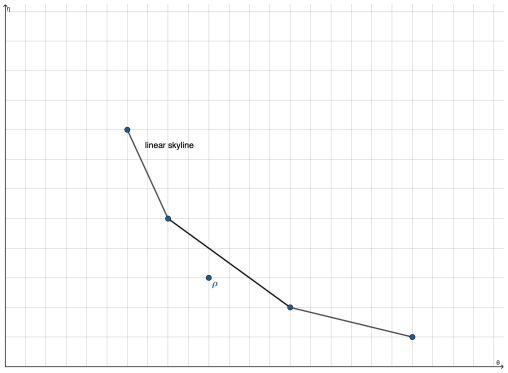


Figure 4.3: LRS and ρ before applying Definition 6.

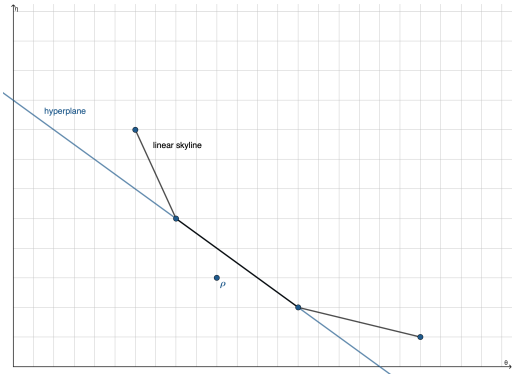


Figure 4.4: LRS after determining a hyperplane using the nearest neighbors.

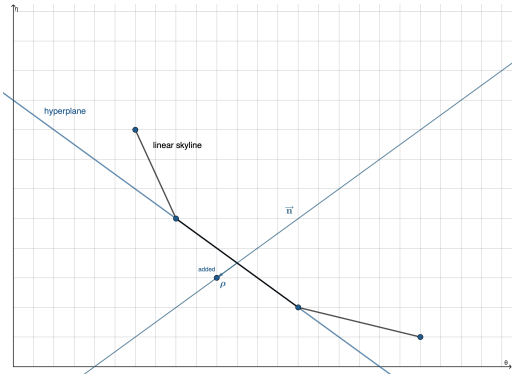


Figure 4.5: In this case ρ belongs to the LRS since $\overrightarrow{n}$ points into the same direction.

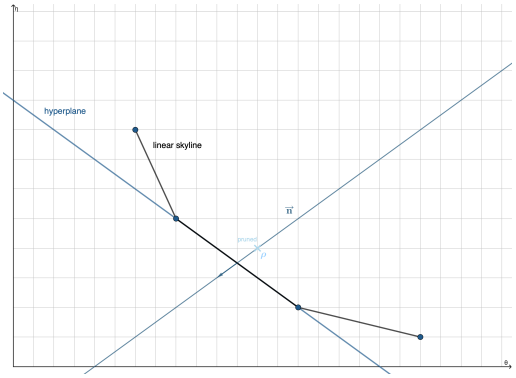


Figure 4.6: In this case ρ does not belong to the LRS since $\overrightarrow{n}$ points into the opposite direction.

Algorithm 4.1: Method to determine if ρ belongs to an LRS.

Data: Let Υ be a set of routes with a 2-dimensional cost vector, ρ a new route, where $\rho \in \Upsilon$, and Σ be the LRS of Υ with $\Sigma \subseteq \Upsilon$.
Result: Σ' containing ρ, otherwise Σ if ρ does not belong to it.

$\Sigma' \leftarrow \Sigma$;
ρ', and ρ'' $\leftarrow$ determine the two nearest neighbors of ρ;
is_part $\leftarrow$ apply Definition 6 using ρ, ρ', and ρ'';
if *is_part* **then**
  $\Sigma' \leftarrow \Sigma \cup \rho$;
end
return Σ';

4.1.3 Linear Preference Boundary

The motivation for preference-bound routing is to reduce the possible result skylines even further since the number of Pareto-optimal solutions increases exponentially with the number of cost criteria [2]. Furthermore, the result set is often too big to be used because confronting a user with many alternatives is often not very helpful.
To address this problem, we introduce Linear Preference Boundaries (LPBs).

We start with the introduction of Definition 7.

Definition 7 (Augmented Route (AR))**.** An Augmented Route (AR) is a route from a (given) source to a (given) destination, where not only the route but also the Fuel-Time-trade-off (short FT-trade-off) for each traversed edge is specified.
Route augmentation refers to selecting a specific trade-off for each edge.

In other words, we extend the knowledge base about a route with a trade-off to be able to consider preferences.
Extending a route to an AR, several facts arise out of Definition 7:

Remark 2. A linear preference in the FT-space can be captured by a single preference coefficient α, as follows $\lambda(\rho) = \alpha \cdot \mathrm{F} + (1 - \alpha) \cdot \mathrm{T}$, where $\lambda(\cdot)$ is the cost function, ρ is an AR, F is the total (summed) fuel consumption across ρ and T is the total (summed) time across ρ.

Remark 3. Every result AR, i.e., every AR in the LRS for the given source-destination pair, corresponds to a range of α values, i.e., the α-Range. The α-Range for all Augmented Routes (ARs) in the LRS partitions the domain of α in the range [0.0, 1.0].

Remark 4. AR's α-Range is determined by its edge's FT-trade-offs and other (competing) result ARs.

In the following, we prove that every edge of a result AR constraints all the trade-offs chosen for every other edge along ρ, as stated in Theorem 1.

Theorem 1 (Constrained FT-trade-offs)**.** Consider a result AR ρ. Focusing on a single edge ϵ in ρ, the FT-trade-off chosen for that edge constrains the trade-off chosen for every other edge along ρ.

Proof. Let us consider the AR ρ fixed, except for the trade-off chosen for ϵ, i.e., consider as different options only the different trade-offs along ϵ.
Assume that FT-ratio sorts the trade-offs in ϵ and that ρ takes the ith out of them, i.e., tf_i, where $i \in \mathbb{N}$.
The cost function $\lambda(\cdot)$ is additive w.r.t. the edges and $\lambda(\rho) = \lambda(\rho') + \lambda(\epsilon)$, where ρ' is the AR except ϵ, i.e., the fixed-trade-off part of ρ. It holds that $\lambda(\text{tf}_i) \leq \lambda(\text{tf}_{i-1})$ and $\lambda(\text{tf}_i) \leq \lambda(\text{tf}_{i+1})$ for every α in the activation range of ρ since tf_i belongs to ϵ. Both inequalities construct a range for α, i.e., [L, R].
Let us now consider any possible pair of trade-offs (tf, tf') in any edge ϵ' in ρ.
If $\forall\ \alpha \in$ [L, R]: $\lambda(\text{tf}) \leq \lambda(\text{tf'})$, then tf' cannot be chosen for ϵ' in ρ.
Hence, we can formally say that trade-off tf' is dominated by tf for edge ϵ'. □

Moreover, the following facts follow directly from Theorem 1:

Remark 5. Specifying the α-Range [L, R] is possible even without knowing the exact α-Range of ρ. Further, proof by contradiction shows that the α-Range is a subset of [L, R].

Remark 6. The intersection of the α-Ranges $[\text{L}_i, \text{R}_i]$, where $i \in \mathbb{N}$, for each edge along ρ is not necessarily the α-Range of ρ. The reason is that the intersection specifies the α-Values for which ρ outscores alternative augmentations along the same route as ρ, i.e., trade-off choices along the same sequence of edges. However, it needs more information to compare ρ with alternative routes.

Remark 7. Following out of Remarks 5 and 6, the intersection of [L, R] ranges across the edges of the AR includes the α-Range of a result AR.

4.1.4 Computing LPBs

To extend the current pruning strategies provided by the ARSC algorithm, we introduced the concept of LPBs in the previous section. In the following, we discuss and justify the assumptions we chose to provide preference-bound pruning.
To incorporate α-Ranges into the ARSC algorithm, we need to change how skylines are calculated. The ARSC algorithm determines for each of the d dimensions a conventional skyline, where $d \in \mathbb{N}$. Further, it uses the lower-bound approximation to prune faster routes. Instead of using a conventional skyline where routes are optimal under some monotonic cost function, we use LRSs to exclude those routes that are not optimal under a linear combination. Finally, we determine the α-Range for a route while considering all possible competitive routes since those routes could dominate each other. To obtain α-Ranges we solve the linear equation system made out of Remark 2.

Remark 8 shows the formula to calculate an α-Range.

Remark 8 (Linear Preference Boundary (LPB)). Let Υ be a set of routes with a 2-dimensional cost vector, ρ, $\rho' \in \Upsilon$ with ρ and ρ' the ith and $i+1$th route of the LRS, where $i \in \mathbb{N}$, and let $\lambda(\cdot)$: $\Upsilon \mapsto \mathbb{R}_{\geq 0}$ be a 2-dimensional cost function. A route's α-Range is determined by:

$$\frac{\lambda_1(\rho') - \lambda_1(\rho)}{\lambda_0(\rho) - \lambda_1(\rho) - \lambda_0(\rho') + \lambda_1(\rho')} .$$

To correctly calculate α-Ranges we need routes that are optimal under linear combinations and follow a partial ordering. We need the first constraint for using Remark 8 otherwise in the case of two non-linear route combinations, Remark 8 could divide by zero since the denominator would be zero, and our definition would not be defined. This fact results from the definition of LRSs. Moreover, we require a partial ordering on linear routes to determine proper intervals. The partial ordering sorts all routes either decreasingly by the first or increasingly by the last dimension. Otherwise, we would be unable to build a proper interval for the LRS because to calculate a sub-interval, we need the ith and $i+1$th node of the LRS, where $i \in \mathbb{N}$. Further, a scattering of the α-Range would lead to an improper pruning, and routes that should not longer belong to the set of possible candidates would remain.

We conclude this section by providing the Pseudocode 4.2 of our method for determining α-Ranges.

Algorithm 4.2: Method to determine α-Ranges for an LRS.

Data: Let Υ be a set of routes with a 2-dimensional cost vector and Σ be the LRS of Υ with $\Sigma \subseteq \Upsilon$.
Result: Δ containing the α-Ranges of Σ.

```
1 sort Σ according to one dimension;
2 Δ ← ∅;
3 foreach succeeding ρ, ρ' ∈ Υ do
4 |   α-Range ← apply Remark 8 for ρ and ρ';
5 |   Δ ← Δ ∪ α-Range;
6 end
7 return Δ;
```

4.1.5 Linear Preference Boundary Skyline Query

Finally, after motivating and introducing the concept of LRSs and LPBs, we introduce our extension to the ARSC algorithm, utilizing α-Ranges to include preference-bound routing. In addition, we extend the basic algorithm with the possibility to use either single- or multi-edged graphs (see Section 6.2). Finally, we explain how our approach calculates LRSs using LPBs.

As introduced in Section 3.4, the ARSC algorithm visits all nodes considered to be part of a result skyline. Moreover, for each node, a list stores all sub-routes ending in that particular node, allowing to prune those sub-routes dominated by other sub-routes ending in the particular node. The rest of the algorithm only takes those sub-routes, expand them with a next hop, and checks for domination. Thereby, it utilizes the lower-bound approximation and the skyline domination criteria before adding the sub-route to the list of sub-routes ending in the particular node and the node to the list of nodes that need to be visited further.
We extend the pruning strategies by adding the LPBs as a further filter step, elevating the skyline to an LRS. To utilize LPBs, we first check if a sub-route is part of the LRS and filter those routes that do not fulfill this filter step. Otherwise, a sub-route is not optimal under a linear combination and cannot be part of the calculation of LPBs (see Section 4.1.4). To filter those routes, we apply Definition 6, introduced in Section 4.1.2. However, we apply this filtering step only when we want to add a new sub-route into the set of result skylines or the list of sub-routes ending in a particular node while only requiring the nearest neighbors of this route for calculations.
Further, we calculate for each sub-route the specific α-Range by taking all competitive sub-routes into account. To calculate the α-Ranges for all sub-routes, we use Remark 8 (see Section 4.1.4) and apply it to all pairs of adjacent sub-routes. As previously mentioned, the list of all sub-routes has to follow a partial ordering. After calculating those α-Ranges, we use them to check if a sub-route is within the preferred range of either the given α-Range determined by the user or within the intersection of the prefix sub-route. We apply this filter step before the ARSC algorithm usually uses the lower-bound approximation to calculate the approximated costs to the destination before we want to add a new sub-route into the set of result skylines or the list of sub-routes ending in a particular node. By doing so, we can already prune those routes that do not share a common α-Range.
We incorporate the ability to use single- or multi-edged graphs. However, using only single-edged graphs, the rest of the algorithm remains the same. In the case of multi-edged graphs, we provide a further filter step for the LRS and calculate for all sub-edged routes the α-Ranges to obtain the LPBs. Since those sub-edged routes that do not share a common intersection of the α-Range of the prefix sub-route are already no longer within the LPBs.

Pseudocode 4.3 and Figure 4.7, conclude this section by visualizing the above-mentioned approach.

Algorithm 4.3: The ARSC algorithm utilizing LPBS-Qs.

Data: Let $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ be a MAG, $\mathbb{L}$ be the Lipschitz embedding, and u, v the source and destination node, where u, $v \in \mathbb{V}$.
Result: Σ containing all non-dominated routes of $\mathbb{G}$.

```
1  Q_V ← {u};
2  Σ ← ∅;
3  while Q_V ≠ ∅ do
4      w ← Q_V;
5      α-Ranges ← determine the α-Ranges for w.sub_routes;
6      foreach ρ ∈ w.sub_routes do
7          if α-Range is not within the LPB then
8              remove ρ from w.sub_routes;
9          else
10             determine the lower-bound approximation for ρ using L;
11             if approximation is dominated by Σ then
12                 remove ρ from w.sub_routes;
13             else
14                 ρ' ← expand ρ;
15                 foreach ρ'' ∈ ρ' do
16                     if ρ''.v == v and is not dominated then
17                         remove all routes that ρ'' dominates;
18                         Σ ← Σ ∪ {ρ''};
19                         α-Ranges ← determine the α-Ranges for Σ;
20                         remove all routes that are not within the LPB;
21                     else
22                         Q_V ← Q_V ∪ {ρ''.v};
23                         if ρ'' is not dominated by ρ''.v.sub_routes then
24                             remove all routes that ρ'' dominates;
25                             ρ''.v.sub_routes ← ρ''.v.sub_routes ∪ {ρ''};
26                         end
27                     end
28                 end
29             end
30         end
31     end
32 end
33 return Σ;
```

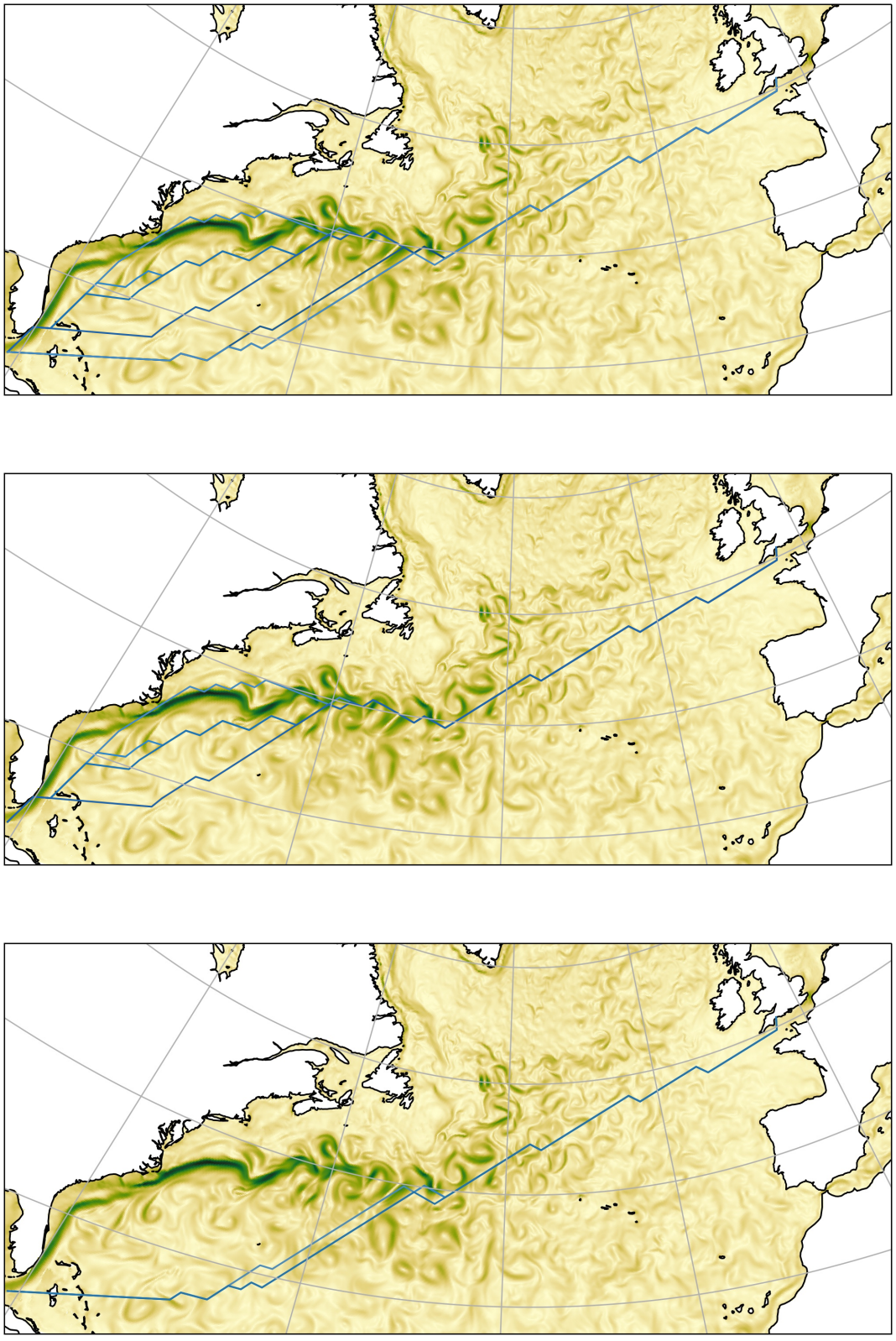


Figure 4.7: LRS (blue lines) after filtering with different α-Ranges.

4.2 Implementation

In the following, we explain how we implemented LPBS-Qs in the context of the ARSC algorithm. Furthermore, we discuss the problems that arose while implementing our approach, i.e., mapping road networks to ocean networks (see Sections 4.2.1 and 4.2.2). Conclusive, we mention our Skyline-routing framework, which contains our implementation (see Section 4.2.3).

Our implementation and the framework is part of the official repository of the working group *Archaeoinformatics - Data Science Research Group*[1] at the University of Kiel.

We start with a discussion of how we have achieved a mapping of road networks to ocean networks without having an underlying graph structure.

4.2.1 Discretizing the ocean

Defining a graph in the context of road networks is easily achievable since edges and nodes are given implicitly by streets and intersections.
However, such information is unavailable in ocean networks since streets and intersections are absent. Moreover, a shipping route is not bound by directions since the shortest route would take the direct line between the source and the destination. Therefore, a different approach is necessary to utilize graphs for ocean networks.
Concerning graphs for ocean routing comes with several problems. The biggest problem arises while mapping a sphere on a plane while wanting to preserve most properties. This problem, called map projection, is part of several fields of pure mathematics, e.g., differential and projective geometry and manifolds. However, map projection refers particularly to a cartographic projection, and the study of map projections is mainly about the characterization of their distortions. Since all projections of a sphere on a plane distort the surface. It depends on the map's purpose, some distortions are acceptable, and others are not. Therefore, different map projections of a sphere-like body exist.

[1] https://git.informatik.uni-kiel.de/ag-ai

4.2.2 Hexagonal hierarchical geospatial indexing system

Sahr et al. [25] proposed that analysis of location data can be done by bucketing locations using a regular grid, providing the ability to measure differences between cells. Thereby, the cell shape of that grid system is an important consideration. However, considering triangles, squares, or hexagons as grid filters, different problems arise, i.e., the distance and the number of neighbors. Triangles and squares have neighbors with different distances. Thereby, only hexagons have an equidistant to all of their neighbors.
Further, the number of neighbors is relatively high, with triangles having 12, squares having eight, and hexagons having only six neighbors. Therefore, hexagons are optimally space-filling. The margin of error is, on average, more minor using polygon ties, i.e., hexagon tiles, than it would be with square tiles. However, using hexagons comes with a drawback since hexagons do not cleanly subdivide into seven finer hexagons. Therefore, alternating the orientation of grids can be necessary to obtain a better approximation for the subdivision into seven smaller cells.

To use graphs for ocean networks, we use H3, a geospatial indexing system, discretizing a global grid system [26] consisting of a multi-precision hexagonal tiling of a sphere with hierarchical indexes. First, it creates a hexagonal grid system on the planar faces of a sphere-circumscribed icosahedron. Then the grid cells are projected to the surface of the sphere using an inverse face-centered polyhedral gnomonic projection, as shown in Figures 4.8 and 4.9.
However, it is only possible to tile the icosahedron partially with hexagons. Therefore, each H3 resolution contains exactly 12 pentagons.
Furthermore, H3 provides various functions to convert and work with latitude and longitude coordinates, allowing us to use boundary geometry and enabling the concept of neighbors.

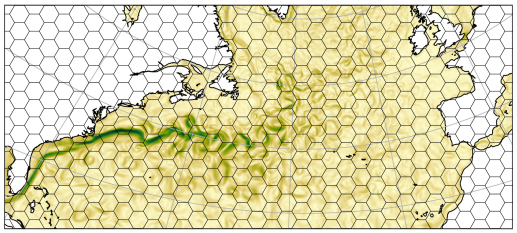


Figure 4.8: North Atlantic covert with a roughly H3 resolution.

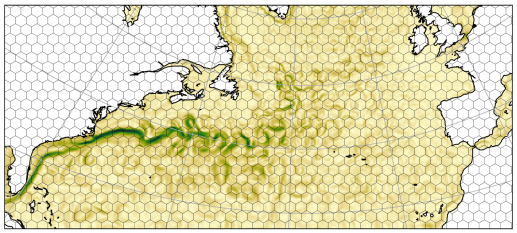


Figure 4.9: North Atlantic covert with a more precise H3 resolution.

4.2.3 Skyline-routing Framework

Our implementation is part of the Skyline-routing framework that comprise several other algorithms. It provides a variety of different methods to utilize different strategies for using skyline routing. Further, it provides functionalities to adjust, verify, and visualize the data.

One cycle of using our extension LPBS-Qs looks as follows:

1. First, we have to write a .config-file to tell the framework which algorithm we want to execute. Furthermore, we set all parameters for executing, i.e., the H3 resolution, the source and destination node, and the start day. More parameters are optional.
2. After passing all parameters, the Lipschitz embedding is generated.
3. Following, the ARSC algorithm determines the LRS applying LPBS-Qs.
4. Conclusive, the results can be visualized.

Chapter 5
Experiments

In the following chapter, we discuss the experimental results of using LPBS-Qs in the context of the ARSC algorithm. Thereby, we analyze our theoretical concepts in real-world scenarios using actual data to evaluate our approach's behavior and if the expected gains in efficiency are achievable.

We begin by mentioning the actual data, the used parameters, and the setup for evaluating our experiments (see Section 5.1). Further, we discuss the particular experiments in detail and conclude with our results (see Section 5.2).

5.1 Experimental Setup

This section discusses the used data, the used parameters, and the setup for our experiments. To be comparable and reproducible, we use a set of basic settings for all experiments, which we also provide in the following.

As described in Section 4.2.3, we need a dataset that provides the current, the waves, and the wind speed information. For this purpose, we used the dataset provided by Mercator Ocean (International) [27]. It provides data for the North Atlantic while covering an area of about 1.4×10^7 m^2. Furthermore, it contains measurements for a grid of latitude and longitude coordinates where each latitude and longitude coordinate pair differs by 0.5°. One of the benefits of this dataset is that it comes with a forecast for those attributes, allowing to add temporal shifts in our calculations without the need for interpolations. Therefore we could use more actual currents, waves, and wind speeds, allowing further a more accurate calculation of costs since we could take advantage of the provided forecast. We used for our calculations a forecast that covered one month, i.e., from December 1st to December 31st, 2020.

Our graphs and the corresponding Lipschitz embeddings have different sizes regarding the number of nodes and edges since we use different H3 resolutions.

Table 5.1 lists all information about the number of nodes, edges, and the average edge length.

Meta information about the H3 graphs			
H3 resolution	number of nodes	number of edges	average edge length in km
0	15	44	1.107×10^3
1	57	236	4.186×10^2
2	288	1466	1.582×10^2
3	1760	9838	5.981×10^1

Table 5.1: A listing of all H3 graphs and their number of nodes, edges, and the average edge length used during evaluation.

While higher H3 resolutions increase the accuracy of a route's actual costs, it expands the graph and decreases the preprocessing and calculation process. Therefore, we fix the number of reference nodes for the Lipschitz embedding to two randomly sampled nodes out of the available nodes and use only H3 resolutions up to and including three. Higher H3 resolutions would not be practicable in our case. For example, building a Lipschitz embedding for a multi-edged graph with four outgoing edges for each node with two reference nodes and an H3 resolution of three takes about 44 h.
Furthermore, as introduced in Section 4.1.5, we extended the ARSC algorithm using multi-edged graphs. To further reduce long runtimes, we also fix the number of edges to four outgoing edges for each node for our experiments.
Since it would be possible to add the costs for an edge directly in the graph, we determine the costs per edge while taking the edge to incorporate more actual current, waves, and wind speed information. Further, it allows us to reduce the size of the materialized graph.
Each of our experiments consists of three phases: applying the data, graph, and Lipschitz embedding, running our real-world scenarios, and evaluating the behavior. Initially, we generate our graph with an H3 resolution up to and including three. Upon this, we create the Lipschitz embedding by using two randomly sampled reference nodes while running Dijkstra's [15] shortest-path algorithm. Next, we execute our approach using the H3 graph, Lipschitz embedding, and preconfigured parameters. Finally, we evaluate our approach using built-in functions of the Skyline-routing framework and a custom script that uses the collected data to compare and determine the overall efficiency.

Before we discuss the experimental results of using LPBS-Qs in the context of the ARSC algorithm, we conclude with an overview of the environment we used for evaluation. As mentioned, we run all our experiments under the same conditions to be reproducible, which includes the software, i.e., the used Python™ and libraries versions, and the target machine.

The following tables, Table 5.2 and Table 5.3, summarize all this information.

Hardware specifications	
component	specification
System on a chip (SoC)	Apple M1 Max chip with 3.2 GHz
Random-access memory (RAM)	32 GB unified memory
Storage	1 TB Solid-state drive (SSD)
Wireless	802.11ax Wi-Fi 6 and Bluetooth 5.0
Operating System (OS)	macOS Ventura (version: 13.0.1)

Table 5.2: Hardware specifications for the evaluation.

Software specifications	
software/library	version
Skyline-routing framework	1.0
Python™	3.10.0 (build: hdfd78df_5)
geopy	2.2.0 (build: pyhd8ed1ab_0)
h3-py	3.7.3 (build: py310hba3363e_3)
networkx	2.5 (build: py_0)
numpy	1.21.5 (build: py310hdcd3fac_1)
numpy-base	1.21.5 (build: py310hfd2de13_1)
pandas	1.4.1 (build: py310he9d5cce_1)
xarray	0.16.1 (build: py_0)

Table 5.3: Software specifications for the evaluation.

5.2 Results

In the following, we want to discuss the experimental results. Thereby, we try to answer the following research questions:

- How do LPBS-Qs impact the overall runtime (see Section 5.2.1)?
- What impact does the length and position of an α-Range have on the behavior of our approach? - Further, is it worth spending more time to find a suitable α-Range as a preprocessing step to improve finding more fitting LRSs (see Section 5.2.2)?

We start with the first research question.

5.2.1 How do LPBS-Qs impact the overall runtime?

To determine the efficiency of our approach, we analyze the overall runtime using different H3 resolutions and outgoing edges from a node. We create different .config-files for real-world scenarios, including source and destination nodes, start days, and route lengths. Furthermore, we use different α-Ranges, i.e., different ranges of different lengths, in comparison to the second half, where we use the same interval at different positions.

Our analysis shows that using LPBS-Qs in the context of the ARSC algorithm can reduce the overall runtime. Furthermore, it reduces the number of filter steps since we do not consider all routes anymore because most routes will be outside the α-Range and, therefore, not part of the possible search space. However, this improvement depends on the interval used. Larger intervals will reduce the gained benefits until the runtime equals the average runtime of the ARSC algorithm, as shown in Figures 5.1 and 5.2.
We can see that using the whole interval, i.e., the α-Range [0.0, 1.0], forces our approach to consider all routes since all attributes are equally preferred. Furthermore, using the whole α-Range decreases the runtime of the ARSC algorithm since determining the α-Range for each node of the MAG and the filter step of the LRS, as shown in Sections 4.1.2 and 4.1.4, takes more time than the naive domination filter step.
Moreover, the improved runtime depends strongly on higher H3 resolutions. If we compare Figures 5.1, 5.3, and 5.4, we can see that the α-Range does not impact the runtime in smaller graphs (see Figure 5.1). Graphs built using a smaller H3 resolution have fewer nodes and edges, as shown in Table 5.1, and therefore smaller search spaces. As a result, finding routes takes less time, even with larger α-Ranges. Since a naive approach that first builds all routes and then prunes dominated routes would also take less time.

However, in larger graphs, it makes sense to use α-Ranges that do not consider all possible routes and to reduce the interval (see Figures 5.3 and 5.4) while gaining better runtime compared to the ARSC algorithm without LPB.
The most significant difference is observable when using α-Ranges in multi-edged graphs, as shown in Figures 5.5 and 5.6. The runtime of the ARSC algorithm finding Pareto-optimal routes in graphs with many outgoing edges from a node can drastically be improved using α-Ranges since a specific α-Range allows prune routes faster without unnecessarily extending them. Further, our approach can converge faster in the direction of Pareto-optimal routes since the A$\star$ (A-star) search algorithm finds more suitable routes to the destination.

An interesting side note is that the filter step to pruning those routes that are non-linear is unnecessary when utilizing only non-intersecting paths. Further, the implicit α-Range, i.e., an α-Range determined by the sub-route, is enough for pruning because Theorem 1 holds and an extended sub-route's α-Range cannot change the prefix sub-route preference. In other words, the preference does not change. For example, if someone is interested in finding the shortest route while focusing on time, no one will change the preference to minimize fuel consumption while exploring the MAG. This behavior would result in an unuseful LRS. However, this fact remains only a corner case since to ensure that the implicit α-Range is enough for pruning, we need to ensure that no other sub-route leads to the same destination and need to know how the sub-route proceed. Otherwise we would calculate the wrong LRSs since we cannot ensure that we collect only linear Pareto-optimal sub-routes.

Overall, using α-Ranges has an impact on the runtime. Taking all preferences into account, i.e., the α-Range [0.0, 1.0], the runtime deteriorates. However, reducing the length of the α-Range, improves the runtime and enhanced the overall runtime of the ARSC algorithm, even with the overhead of creating and maintaining an LRS.

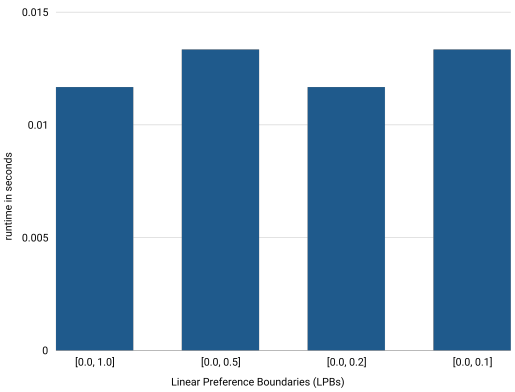


Figure 5.1: Runtimes of a H3 graph with a roughly resolution using different lengths of α-Ranges.

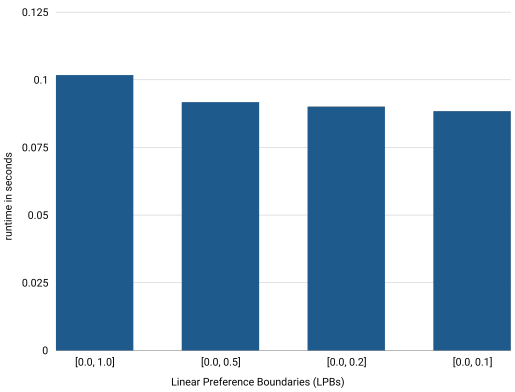


Figure 5.2: Runtimes of a H3 graph with a less roughly resolution using different lengths of α-Ranges.

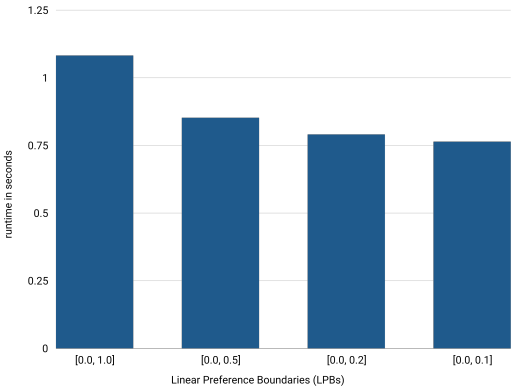


Figure 5.3: Runtimes of a H3 graph with a more precise resolution using different lengths of α-Ranges.

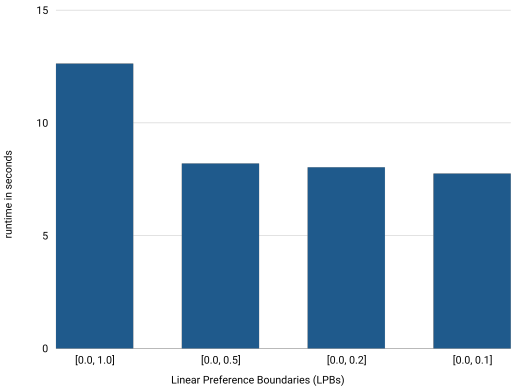


Figure 5.4: Runtimes of a H3 graph with a precise resolution using different lengths of α-Ranges.

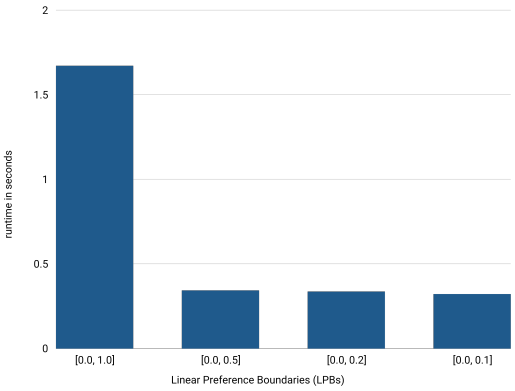


Figure 5.5: Runtimes of a multi-edged graph with a roughly resolution using different lengths of α-Ranges.

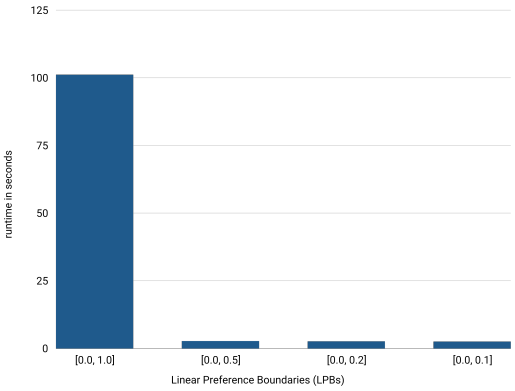


Figure 5.6: Runtimes of a multi-edged graph with a precise resolution using different lengths of α-Ranges.

5.2.2 What impact does the length and position have?

During our analysis, we could see that the length and the position impact the runtime of our approach. The same α-Range, i.e., the same length, at different positions results in different runtimes, as shown in Figure 5.7. In all experiments, the most extended runtimes appear at the end of the α-Range. This behavior is because most of the preference collapses at the end of a route, resulting in longer runtimes.
Moreover, different lengths of α-Ranges results in different runtimes as well. As mentioned in the previous section using the whole α-Range is equal to not using any preference and only applying an LRS. Furthermore, different lengths split the search space into different-sized sub-spaces resulting in different runtimes. Therefore, smaller α-Ranges are faster than larger ones. Nevertheless, this behavior also depends on the size of the graph, as already discussed. Especially using multi-edged graphs suffer from inconvenient chosen α-Ranges since the runtime can increase by several times compared to other α-Ranges, as shown in Figure 5.8. Therefore, finding good α-Ranges that fit the preference could make sense. However, determining suitable α-Ranges is quite challenging since no proper mapping between α-Ranges and preferences exists. During our experiments, we could figure out that smaller α-Ranges preferred those routes that minimized fuel consumption, while larger α-Ranges preferred those routes that reduced the needed time. Since this mapping is not general and we cannot provide a promising strategy besides running a couple of tests beforehand, we have no evidence that determining a well-fitted α-Range leads to a well-fitted LRS. Moreover, the abovementioned behavior could be dataset-specific and have no impact on the runtime.

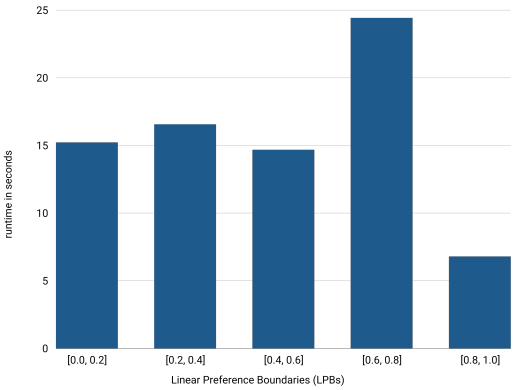


Figure 5.7: This result shows that the same LPB at different positions can result in different runtimes.

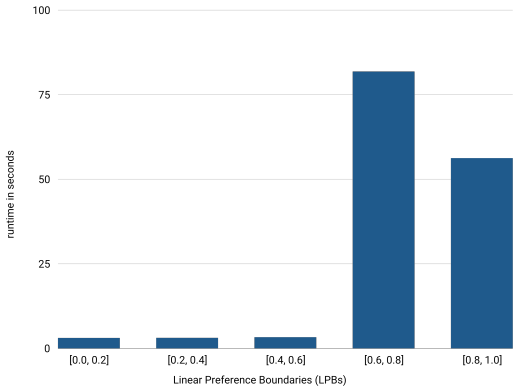


Figure 5.8: This result shows how the runtimes of multi-edged graphs suffer using inconvenient α-Ranges.

Chapter 6
Conclusion and Improvements

In this last chapter, we summarize our contributions (see Section 6.1) while discussing further improvements and outlining the possible further direction for research (see Section 6.2). First, we begin with an overall summary.

6.1 Conclusion

Since shipping transports an overwhelming majority of goods, ocean shipping is the backbone of global trade and an integral part of the supply chain for most industries while contributing to climate change and global warming [1]. Recently, this problem has become more relevant in economy and politics, e.g., by developing new methods to reduce the influence of greenhouse gases and new laws limiting the amount of released CO_2 through carbon taxes.

This thesis provided a method to find the best-fitting Pareto-optimal shipping routes, leveraging the ARSC algorithm to utilize LPBs while considering environmental conditions, such as currents, waves, and wind, providing a real-world approach. Thus, the user can select from various routes best fitting their preference, e.g., the most climate-friendly route.
Since other methods, e.g., shortest-path algorithms, do not allow incorporating unknown external constraints, e.g., safety paths or statutory provisions, we used LRS-Qs for routing to find the best-fitting Pareto-optimal shipping routes. Thereby, we could show that our approach improves the overall runtime in different real-world scenarios while improving the importance of the result since only those routes remain that best fit the user preference. Further, we could show that LPBs tremendously shrink the possible search space in large multi-edged graphs since we were able to prune candidates faster.

During the thesis, we provided the related work while classifying our proposed work, introducing different approaches for solving associated problems while presenting recent studies on LRSs (see Chapter 2). Further, we discussed the background knowledge and the terms and definitions needed to understand our approach (see Chapter 3). Then, we introduced our primary contribution, preference-bound pruning, in the context of route-skyline queries. Moreover, we proposed theoretically and empirically a new approach to determine an LRS in dynamic and uncertain search spaces while using LPBS-Qs as a pruning filter for LRSs. We showed why we had to evolve a different strategy to enable linear skylines. Afterward, we presented our extension for the ARSC algorithm for efficiently computing multi-attribute LRSs in MAGs using LPBs to prune routes faster, allowing single- and multi-edge graphs. Moreover, we discussed the implementation of our approach, the requirements we had to choose from, the problems that arose while implementing LPBS-Qs, and the Skyline-routing framework that contains our implementation. Next, we mentioned how we enabled the usage of graphs in ocean routing utilizing H3 (see Chapter 4) and concluded with an analysis of LPBS-Qs and a discussion of how LPBS-Qs impact the overall runtime and what impact the length and position of an α-Range have on the behavior of our approach (see Chapter 5).

6.2 Improvements

The main drawback of our approach is that we cannot incorporate dynamic changes in the underlying data. However, only an adaptable algorithm would make sense in a real-world application. In case, for example, the current, the waves, or the wind speed changes, our approach would not be able to consider these changes. Furthermore, delays while traveling to the destination would increase the costs and greenhouse gas emissions, making the usage of our approach redundant. A completely new calculation would be required to incorporate these changes, including rebuilding the Lipschitz embedding since the shortest path would not necessarily be valid. Due to the relatively long runtime of the preprocessing step, it does not make sense to rebuild the complete preprocessing when the measured data changes. However, utilizing the new data is hard since these changes can belong to routes that are no longer part of the set of LRSs but normally would if the changes were available at the beginning of the calculation. Worse, a route that is no longer valid could remain part of the LRS even though the route would no longer fulfill the requirements stated in Chapter 4.

One possibility would be to divide the graph into different regions before the preprocessing calculates the Lipschitz embedding. With independent areas, discarding the whole Lipschitz embedding would no longer be necessary while changes occur. We may only need to split those areas into separate Lipschitz embeddings where changes in the ocean data often occur. Therefore, we have to redo the preprocessing in those smaller areas. Since merging the different parts would not be necessary, no further overhead would reduce the runtime gain. However, this could speed up the preprocessing but not change the problem of wrong routes already included in the set of results or pruned by violating the lower-bound approximation. Therefore, to cope with this problem, we probably need to carry a set of already pruned routes along and check them against these changes. However, this would also require to expand those routes further, even though they may not be helpful. Nevertheless, this would still return a smaller subset of an LRS to the user, probably sacrificing all speed improvements making this approach unprofitable.
An alternative could be to incorporate time and split the graph temporally rather than spatially. For example, we could use long-term measurements of ocean data as forecasts of costs in the graph and use that information in a multi-edged regime, taking the best reasonable costs when we visit the node. Of course, while doing so, more shortest-path calculations have to be considered. However, it would improve the approach by adding dynamic costs.

Bibliography

[1] "Ocean shipping worldwide - statistics & facts". (2022), [Online]. Available: `https://www.statista.com/topics/1728/ocean-shipping` (visited on 10/30/2022).

[2] Shekelyan, Michael, Jossé, Gregor, and Schubert, Matthias, "Linear path skylines in multicriteria networks", in *2015 IEEE 31st International Conference on Data Engineering*, 2015, pp. 459–470.

[3] Borzsony, Stephan, Kossmann, Donald, and Stocker, Konrad, "The Skyline operator", in *Proceedings 17th International Conference on Data Engineering*, 2001, pp. 421–430.

[4] Müller-Hannemann, Matthias and Weihe, Karsten, "Pareto Shortest Paths is Often Feasible in Practice", in *Algorithm Engineering*, 2001, pp. 185–197.

[5] Mouratidis, Kyriakos, Li, Keming, and Tang, Bo, "Marrying Top-k with Skyline Queries: Relaxing the Preference Input While Producing Output of Controllable Size", in *Proceedings of the 2021 International Conference on Management of Data*, 2021, pp. 1317–1330.

[6] Chomicki, Jan, Godfrey, Parke, Gryz, Jarek, and Liang, Dongming, "Skyline with presorting", in *Proceedings 19th International Conference on Data Engineering (Cat. No.03CH37405)*, 2003, pp. 717–719.

[7] Morse, Michael, Patel, Jignesh M., and Jagadish, H. V., "Efficient Skyline Computation over Low-Cardinality Domains", in *Proceedings of the 33rd International Conference on Very Large Data Bases*, 2007, pp. 267–278.

[8] Tan, Kian-Lee, Eng, Pin-Kwang, and Ooi, Beng Chin, "Efficient Progressive Skyline Computation", in *Proceedings of the 27th International Conference on Very Large Data Bases*, 2001, pp. 301–310.

[9] Kossmann, Donald, Ramsak, Frank, and Rost, Steffen, "Shooting Stars in the Sky: An Online Algorithm for Skyline Queries", in *Proceedings of the 28th International Conference on Very Large Data Bases*, 2002, pp. 275–286.

[10] Papadias, Dimitris, Tao, Yufei, Fu, Greg, and Seeger, Bernhard, "An Optimal and Progressive Algorithm for Skyline Queries", in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, 2003, pp. 467–478.

[11] Köhler, Ekkehard, Möhring, Rolf H., and Schilling, Heiko, "Acceleration of Shortest Path and Constrained Shortest Path Computation", in *Experimental and Efficient Algorithms*, 2005, pp. 126–138.

[12] Chan, Chee-Yong, Jagadish, H. V., Tan, Kian-Lee, Tung, Anthony K. H., and Zhang, Zhenjie, "Finding k-Dominant Skylines in High Dimensional Space", in *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, 2006, pp. 503–514.

[13] Li, Cuiping, Ooi, Beng Chin, Tung, Anthony K. H., and Wang, Shan, "DADA: A Data Cube for Dominant Relationship Analysis", in *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, 2006, pp. 659–670.

[14] Yuan, Yidong, Lin, Xuemin, Liu, Qing, Wang, Wei, Yu, Jeffrey Xu, and Zhang, Qing, "Efficient Computation of the Skyline Cube", in *Proceedings of the 31st International Conference on Very Large Data Bases*, 2005, pp. 241–252.

[15] Dijkstra, Edsger W., "A Note on Two Problems in Connexion with Graphs", 1959, pp. 269–271.

[16] Kriegel, Hans-Peter, Renz, Matthias, and Schubert, Matthias, "Route skyline queries: A multi-preference path planning approach", in *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, 2010, pp. 261–272.

[17] Stewart, Bradley S. and White, Chelsea C., "Multiobjective A$\star$", 1991, pp. 775–814.

[18] Yang, Bin, Guo, Chenjuan, Jensen, Christian S., Kaul, Manohar, and Shang, Shuo, "Multi-Cost Optimal Route Planning Under Time-Varying Uncertainty", 2013.

[19] Machuca, Enrique and Mandow, Lawrence, "Multiobjective Heuristic Search in Road Maps", 2012, pp. 6435–6445.

[20] Özpeynirci, Özgür and Köksalan, Murat, "An Exact Algorithm for Finding Extreme Supported Nondominated Points of Multiobjective Mixed Integer Programs", 2010, pp. 2302–2315.

[21] Shekelyan, Michael, Jossé, Gregor, Schubert, Matthias, and Kriegel, Hans-Peter, "Linear Path Skyline Computation in Bicriteria Networks", in *Database Systems for Advanced Applications*, 2014, pp. 173–187.

[22] Mote, John, Murthy, Ishwar, and Olson, David L., "A parametric approach to solving bicriterion shortest path problems", 1991, pp. 81–92.

[23] Brumbaugh-Smith, J. and Shier, Douglas R., "An empirical investigation of some bicriterion shortest path algorithms", 1989, pp. 216–224.

[24] Skriver, Anders J.V. and Andersen, Kim Allan, "A label correcting approach for solving bicriterion shortest-path problems", 2000, pp. 507–524.

[25] Sahr, Kevin, White, Denis, and Kimerling, Jon A., "Discrete Global Grid System", 2003, pp. 121–134.

[26] "Uber's H3 - hexagonal hierarchical geospatial indexing system". (2022), [Online]. Available: `https://h3geo.org` (visited on 10/30/2022).

[27] "Mercator Ocean International". (2022), [Online]. Available: `https://www.mercator-ocean.eu/en/` (visited on 10/30/2022).