Optimization Algorithms for the Flight Planning Problem

Dissertation

zur Erlangung des Grades eines
Doktors der Naturwissenschaften

am Fachbereich Mathematik und Informatik
der Freien Universität Berlin

vorgelegt von

Marco Blanco Sandoval

Berlin, 2023

Erstgutachter und Betreuer: Prof. Dr. Ralf Borndörfer

Zweitgutachter: Prof. Dr. Marco Chiarandini

Tag der Disputation: 13. September 2023

Abstract

The Flight Planning Problem is a type of Shortest Path Problem that deals with calculating a minimal-cost flight trajectory between two airports. This trajectory must take into account various types of operative restrictions constraining it both horizontally and vertically. Further factors such as weather (in particular wind), aircraft performance, and fuel on board must be considered. The goal is to minimize the sum of fuel costs, time costs, and overflight costs.

Despite the complexity of this problem, which requires the processing of large amounts of data, real-world applications demand that it be solved within minutes or even seconds. This creates the need for fast solution algorithms yielding high-quality – ideally optimal – solutions. In this thesis, we explore various aspects of the Flight Planning Problem and propose several novel algorithms. Our tests on real-world data show that they significantly improve the state-of-the art. In some cases, they are the first of their kind.

We present a method to reliably underestimate air distance which allows for the creation of an A* algorithm to solve a variant of the Flight Planning Problem that assumes constant altitude. This algorithm, which generalizes the Time-Dependent Shortest Path Problem (TDSPP), is guaranteed to be optimal under assumption of the First-In-First-Out property, and is on average 20 times faster than the Dijkstra algorithm. In a classical TDSPP setting with piecewise-linear travel time functions, our A* outperforms the state-of-the-art approach by a factor of 15.

We also propose a *pruning* technique for significantly reducing the search space of the problem while preserving optimality. We show that a Dijkstra algorithm running on this reduced space is up to 6 times faster than one based on a naive search space reduction. To the best of our knowledge, this is the first work in the literature that presents a highly performant algorithm for solving the Flight Planning Problem with variable altitude, under consideration of not only fuel and time, but also overflight costs.

We also extend our first A* algorithm to a version that runs on the three-dimensional version of the problem by defining a sophisticated potential function based on *idealized vertical profiles*. This potential is proven to be both admissible and consistent under reasonable assumptions, ensuring optimality. Additionally, we introduce two preprocessing techniques to speed up the calculation of the queries and show that our algorithm algorithm outperforms the A* algorithm that previously yielded the best results, resulting in a speed-up of up to 40%.

The importance of overflight costs in flight planning is often overlooked in the literature. We introduce a heuristic that transforms route-based overflight costs into arc-based costs and integrate them into a classical Dijkstra algorithm. The required preprocessing step is fast enough to be practical in applications. A Dijkstra algorithm using the calculated arc costs is faster than the best known exact algorithm by a factor of over 350, with an optimality gap of less than 1%.

Finally, we conduct a polyhedral analysis of the Path Avoiding Forbidden Pairs polytope, motivated by the study of complex traffic restrictions present in the Flight Planning Problem.

Under certain conditions, we are able to provide a complete linear description of the polytope, together with a polynomial-time separation routine.

Acknowledgements

First and foremost, I would like to thank my supervisor, Prof. Dr. Ralf Borndörfer, for giving me the opportunity to work at the Zuse Institute Berlin and in such an exciting topic. He encouraged me to take responsibility and become independent from early on. His ideas and ways of doing research greatly influenced me and my work.

The work of Prof. Marco Chiarandini on flight planning algorithms has greatly influenced and motivated me in my research. I would like to express my sincere gratitude to him for his willingness to participate in the review process of this thesis.

I would like to thank all my coauthors for their collaboration in this research. A special mention goes to Adam Schienle, Pedro Maristany and Nam-Dũng Hoang. We spent countless hours over many years brainstorming and discussing details. Significant parts of this work were obtained in collaboration with them. Furthermore, they made being in our office a very enjoyable experience.

Lufthansa Systems initiated the research collaboration with the Zuse Institute Berlin, which made this thesis possible. Particular thanks go to Anton Kaier and Swen Schlobach, who were always happy to provide data, aeronautical expertise, and research directions.

Finally, I want to thank my family for their support over all these years, especially in the form of childcare. Many thanks to my wife Emma for her patience, encouragement, and sacrifices throughout this process.

Contents

Chapter 1

General Introduction

The growth of air traffic presents a number of challenges for the aviation industry and society as a whole. Despite the severe blow inflicted by the COVID-19 pandemic on the aviation sector, air traffic is expected to exceed pre-crisis levels by 2024 and reach 111% by 2025[Int22]. This growth highlights the need to address the various challenges that come with it, including those of an economical, logistical, and environmental nature.

Economically, the growth of air traffic can put a strain on the resources of both airlines and airports. As more flights are added to the schedule, airlines must invest in more aircraft, staff, and maintenance; while airports must expand and upgrade their facilities to accommodate the increased traffic. The cost of fuel is also a significant factor, as it is one of the largest expenses for airlines. As fuel prices rise, it can significantly impact the bottom line of the industry. The International Air Transport Association (IATA) estimates that the total expenditure of airlines on fuel in 2022 was over USD 221 billion and is projected to rise to USD 229 billion in 2023 [22a]. Furthermore, the growth of air traffic can also lead to increased competition among airlines, which can drive down profits and make it difficult for smaller carriers to survive. The COVID-19 pandemic has caused financial struggles for many airlines, which has made it even harder for them to invest in new equipment and infrastructure to keep up with the growing demand for air travel.

In terms of logistical challenges, the growth of air traffic can make it more difficult for air traffic control systems to manage the flow of planes in the sky and on the ground. This can lead to delays, cancellations and other issues that can be costly for both airlines and passengers. Additionally, the increased air traffic can also lead to a more severe noise pollution around airports, which can be disruptive for local residents. Furthermore, a growing number of flights increases the potential for accidents and incidents. Complex regulations for airspace users also present an operational challenge, as they can make it harder for airlines and air traffic control systems to navigate and coordinate the increasing number of flights.

The environmental challenges of growing air traffic are considerable. Airplanes are a significant source of greenhouse gas emissions, and as air traffic grows, so too does the amount of carbon dioxide and other pollutants that are released into the atmosphere. In 2019, fossil fuel burn in the aviation sector was responsible for 1.7% of all CO_2 emissions [Eur22]. As climate change effects become more and more tangible and awareness increases, the industry is facing growing pressure from both regulators and customers to curb its environmental impact. While the aviation sector is working to reduce its carbon footprint through the development of more efficient aircraft and alternative fuels, there is still a long way to go to address these challenges.

The main topic of this thesis, flight planning, addresses these key challenges: By designing cost-efficient flight plans that aim to reduce fuel consumption, flight time, or overflight charges, airlines can make significant savings, which are especially important given the small profit margins typical in the aviation industry. Additionally, with the increasing growth of air traffic, designing flight routes that comply with all regulatory restrictions, while also being physically accurate, quick to calculate, and meeting the specific needs of the airline, is becoming increasingly complex. Furthermore, by designing flight plans that minimize fuel consumption, airlines can take an important step towards reducing their carbon footprint and addressing the urgent issue of climate change.

This thesis delves into the use of mathematical optimization algorithms for flight planning, with the goal of enhancing the quality of flight routes while considering both the efficiency and computational performance of the algorithms. By utilizing advanced optimization techniques, this research aims to find optimal solutions that balance the various factors involved in flight planning, such as fuel consumption, flight time, compliance with regulations and airline requirements, as well as computational efficiency. The outcome of this research will not only improve the economic and environmental sustainability of flight planning, but also help airlines make more informed decisions.

In this chapter, we will provide a general overview of the Flight Planning Problem, as it serves as the foundation and motivation for the research presented in the subsequent chapters. It is important to note that each chapter is self-contained and can be read independently, but this introduction provides additional context that can aid in understanding the complexities and nuances of the research questions investigated in the main chapters.

1.1 The Flight Planning Problem

In this section, we provide an overview of the key components of the Flight Planning Problem (FPP) that form the foundation for the formal mathematical models detailed in subsequent chapters.

At its core, the FPP aims to determine a feasible flight trajectory that minimizes the sum of fuel costs, time costs, and overflight charges incurred by the flight. The problem is constrained by the conditions known to the airline a few hours before the departure, such as weather conditions, aircraft performance, and regulatory restrictions.

It is important to note that different airlines may have distinct requirements and access to different types of data, making it difficult to establish a universal definition of the FPP. The variant of the FPP presented here has been tailored to encompass a majority of the most relevant mathematical challenges present in its real-world equivalents. We consider it to be a general representation of the problem. However, it should be acknowledged that the precise definition of the problem may vary depending on the specific needs and constraints of different airlines.

The FPP is defined by the following:

Input

- A departure- and a destination airport.
- A departure time, usually a few hours in the future.
- A set of weather forecasts, which predict the wind and temperature conditions at all times and locations relevant for the flight. The content of these forecasts and will be presented in Section 1.1.3, as well as an overview of how this data is handled for flight planning.

- The type of aircraft to be used. Of particular interest is *aircraft performance* data, which is used to compute the fuel burn, elapsed time, and altitude at each point of the flight. Other elements relevant for the optimization are the aircraft's *maximum take-off weight* and *dry operating weight* (the weight of an aircraft carrying no fuel and no passenger load), among others. We will discuss aircraft performance functions in more detail in Section 1.1.1.

- The global Airway Network, as described in Section 1.1.2.

- Traffic constraints that need to be satisfied by the flight trajectory. These can range from temporarily closed airspaces to extremely convoluted constraints that force the inclusion/exclusion of certain points in/from the trajectory during certain time intervals depending on its course. Section 1.1.4 focuses on these restrictions.

- A set of *cost airspaces* and associated overflight cost functions. Overflight costs will be discussed in Section 1.1.5.

- Flight-specific information, possibly known to the airline only shortly before the flight, such as estimated payload (the total weight of passengers and luggage), fuel price, cost per hour of flight, or desired flight speed.

- The initial fuel amount to be loaded before the flight.

Output

- A five-dimensional (latitude, longitude, altitude, time, aircraft weight) trajectory connecting both airports and departing at the given time. The following requirements should be met:

  1. The trajectory conforms to the Airway Network, both horizontally and vertically. See Section 1.1.2.
  2. The trajectory is physically feasible. This means that aircraft performance functions (Section 1.1.1) and weather data (Section 1.1.3) are taken into account, in particular when determining the climb/descent angles and the time/weight dimensions. Also, the initial fuel amount needs to be sufficient to operate the complete flight.
  3. All operative restrictions are satisfied.

Objective Function

The goal is to minimize the sum of the following:

- Fuel costs, obtained as the product of fuel price and fuel consumption (which is itself dependent on the flown trajectory, aircraft performance functions, aircraft weight and weather).

- Overflight costs, which are determined by the intersection of the trajectory with cost airspaces.

- Time costs, which mostly consist of personnel and maintenance costs, and which we assume to be linear in the total flight time.

We will expand on these costs in Section 1.1.5.

1.1.1 Aircraft Performance

Both for economic and for safety reasons, it is crucial for flight planning tools to be able to accurately model the aircraft's movement and calculate its fuel consumption. This is done in practice through the use of *aircraft performance* functions which determine how the aircraft's state changes during the flight. These functions are defined for each of the three main flight phases:

- **Cruise**: The aircraft flies at constant altitude.
- **Climb:** The aircraft's altitude increases.
- **Descend:** The aircraft's altitude decreases.

For example, climbing between two altitudes consumes much more fuel than descending the same altitude range.

In the search algorithms we will present, the following question is relevant: If the aircraft is at a given point (defined by its coordinates and altitude) at a given time and with a given weight, and it flies towards another three-dimensional point in a straight line at constant speed, what is its weight (determined by the amount of fuel burnt) at the second point and how long does it take to get there?

The input time and the coordinates are used for calculating the weather-dependent air distance, see Section 1.1.3. This distance is then passed to a *propagation function* together with the weight, speed, and altitude inputs.

We use this propagation function implicitly in Chapters 2 and 3, and explicitly in Chapter 4. It is based on aircraft performance data, which is provided by the aircraft manufacturer and describes the change of the aircraft's state based on the given parameters.

Throughout this thesis, it suffices to consider this propagation function as a black box. In the chapters where it is needed, more information about it will be given. Nevertheless, the interested reader can find a more detailed but also technical description in Appendix 1.4.

1.1.2 The Airway Network

Partly due to historical reasons, but also as an effort to make air traffic control more manageable, airspace users are required to plan routes in such a way that they conform to a well defined structure. In particular, this means that the horizontal route must adhere to a network and cruising (e.g. flying at constant altitude) is only allowed on a small set of altitudes. To define this formally, we need the following definitions:

Definition 1.1. A *waypoint* is a point on the Earth's surface, defined by its geographical coordinates. Aircraft are only allowed to make turns when flying over a waypoint. Historically, waypoints corresponded to landmarks and radio navigational aides. With the advent of technologies such as GPS, waypoints are nowadays simply defined by arbitrary coordinates. We can think of waypoints as nodes in a directed graph.

Definition 1.2. An *airway segment*, or simply *segment*, is a directed connection between two waypoints, following the shortest straight line (geodesic or *great circle*) between them. Segments have a small number of altitudes (called *flight levels*) on which aircraft are allowed to cruise.[a] In this thesis, we assume that a trajectory may consecutively visit two waypoints only if there exists a segment connecting them. In our setting, segments are modeled as directed arcs.

[a]This concept is called RVSM (Reduced Vertical Separation Minima), see [Sil10]. In most airspaces (in particular in Europe), the most common separation between two consecutive flight levels on which cruise is allowed is 1000ft (304.8m). Another common separation is 300m, used for instance in Central Asia and China.

Definition 1.3. The *Airway Network* is a collection of waypoints and segments spanning over the Earth. The Airway Network can be modeled as a directed graph.

Definition 1.4. We say a flight trajectory *conforms to the Airway Network* if it is horizontally and vertically feasible according to the following requirements:

Horizontal Feasibility

1. At every instant, the aircraft's coordinates either correspond to a waypoint, or are located in the interior of a segment. This means that the horizontal component of the trajectory can be seen as a path on a directed graph.

Vertical Feasibility

1. Every cruise phase's altitude corresponds to one of the flight levels allowed by the segment.
2. Following a cruise phase, a change of altitude (e.g. a climb or descent phase) can only be initiated when the aircraft is overflying a waypoint.
3. Whenever a flight level is reached during a climb phase, it is possible to terminate the climb phase and continue cruising at that altitude. We refer to this procedure as *step climb*.
4. If a waypoint is reached via climbing at an altitude between two flight levels, it is necessary to continue climbing. We refer to this procedure as *transition climb*.
5. Rules 3 and 4 apply analogously for descent. We then speak of *step descent* and *transition descent*.

See Figure 1.1 for an example of the Airway Network over France and a flight route crossing it.

1.1.3 The Impact of Weather

Upper air weather plays a critical role in flight planning and its time-varying nature is a major factor that causes optimal flight trajectories to differ from day to day. Air temperature, for instance, has a direct impact on the performance of aircraft engines, making it more fuel-efficient to fly in cooler air, typically at higher altitudes.

However, wind is the most significant factor to consider in flight planning. A strong headwind or tailwind can greatly affect flight time and fuel consumption. For example, transatlantic flights are heavily impacted by the polar jet stream, a fast-moving air current that travels from west to east across the Atlantic. Eastbound flights often take advantage of this current, while westbound flights tend to avoid it by flying either north or south of it. This results in eastbound flights being shorter by a few hours as compared to westbound flights. In Figure 1.2 we can see the impact of wind on optimal trajectories.

The weather data used for flight planning is usually obtained from weather forecasting services such as WAWFOR (World Aviation Weather Forecast)[22b], which is provided by the German Meteorological Service; or WAFS (World Area Forecast System)[22c], which is made available by the U.S. government through its National Weather Service agency. Each forecast consists of a set of data points corresponding to different coordinates, altitudes, and times. Each data point contains a wind vector and a temperature. See Figure 1.3 for an example.

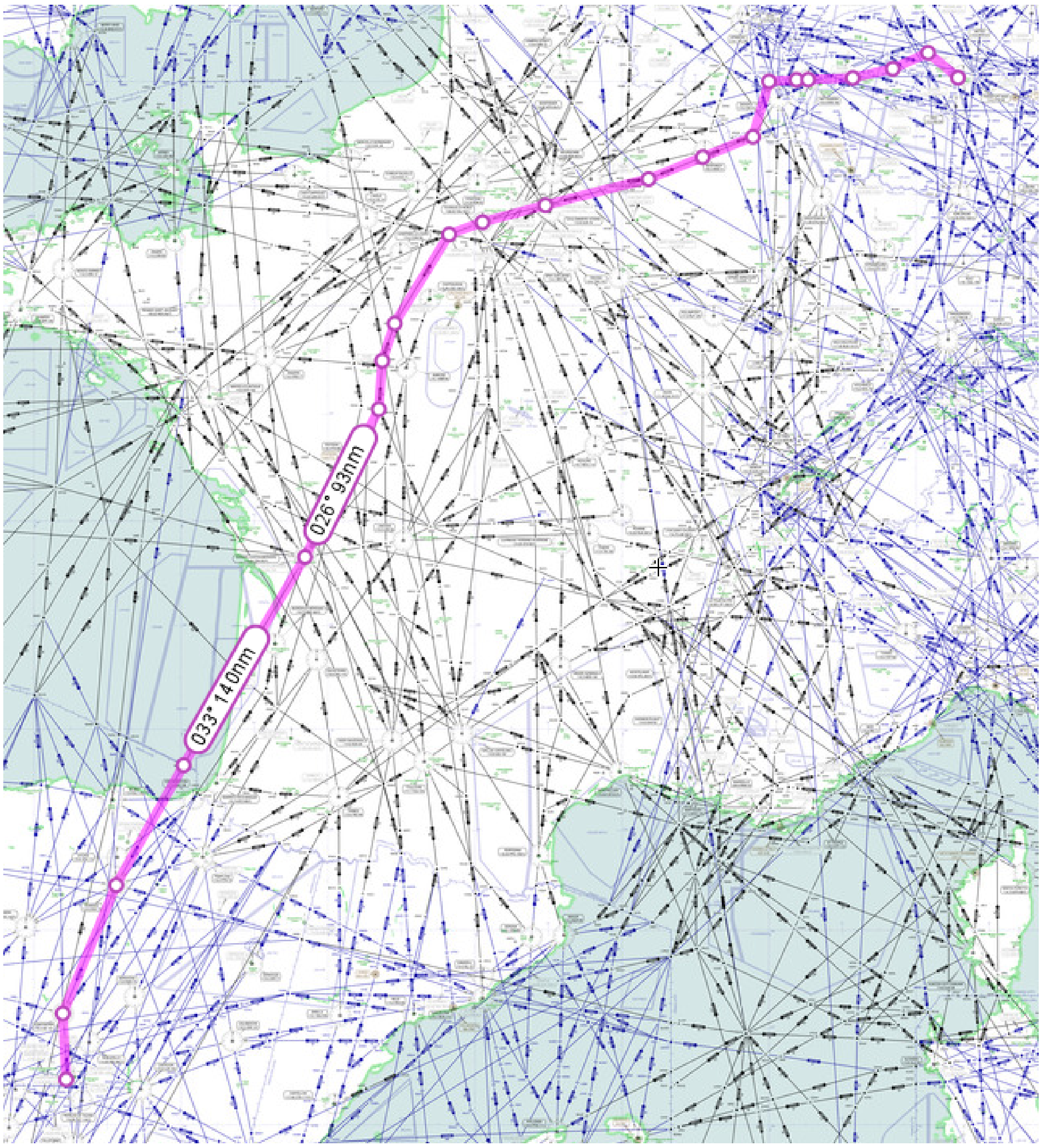


Figure 1.1: Image obtained using skyvector.com. The pink line shows a flight route from Madrid to Frankfurt, following the Airway Network.

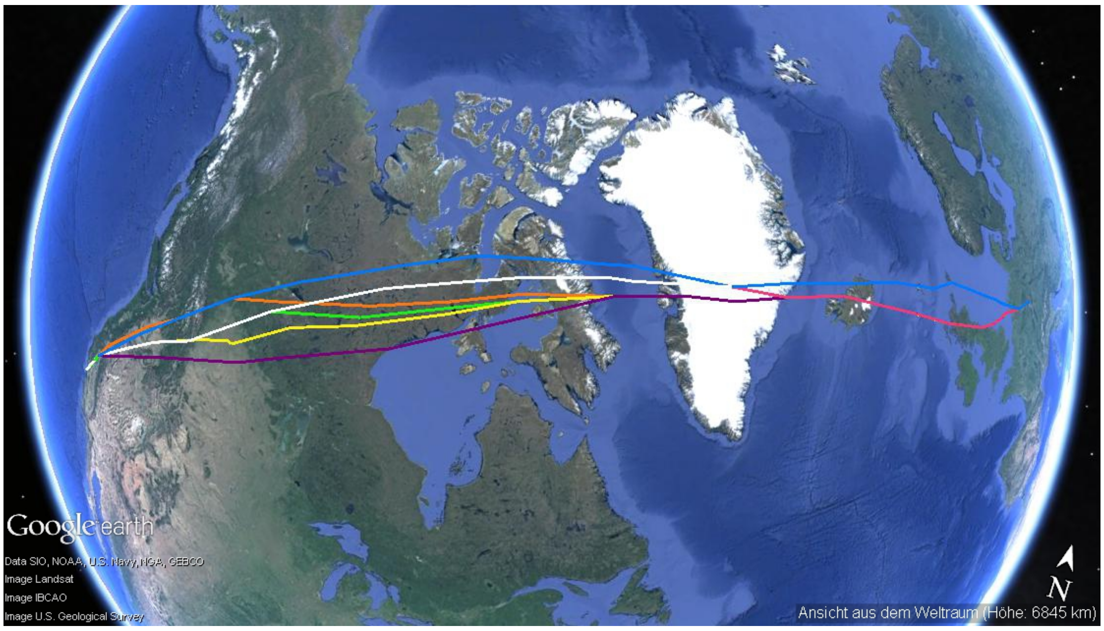

Google earth
Data SIO, NOAA, U.S. Navy, NGA, GEBCO
Image Landsat
Image IBCAO
Image U.S. Geological Survey
N
Ansicht aus dem Weltraum (Höhe: 6845 km)

Figure 1.2: Image produced using Google Earth. Several optimal routes from San Francisco to Frankfurt are shown. All input is equal except for the weather data, which corresponds to different days.

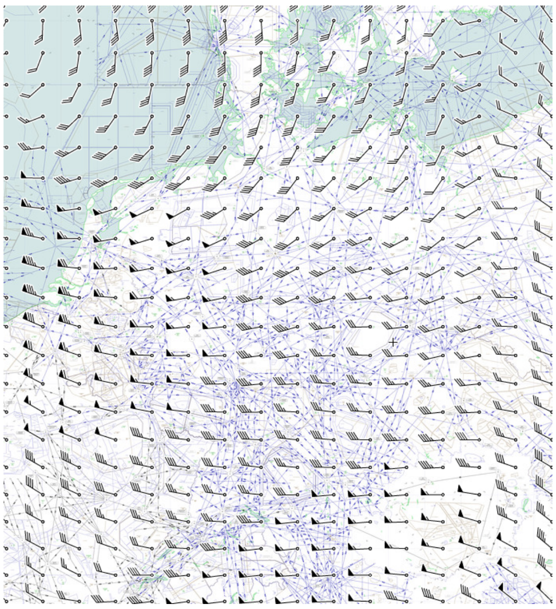


Figure 1.3: Image obtained using skyvector.com. The arrows represent the wind direction and speed over Germany at a given time and altitude.

Since the horizontal component of flight routes consists of a concatenation of great-circle lines (see Section 1.1.2), we are interested in determining the influence of wind not only at specific points in space and time, but on each segment as a whole.

For a given altitude and time, the wind vectors located on grid points as given by the weather forecast are interpolated to obtain a single vector acting on the segment. The bulk of this horizontal interpolation can be done as a preprocessing step, since all relevant coordinates are known a priori. For that reason, the horizontal interpolation is not relevant for this thesis. We will thus assume that, for each segment, we have a matrix of wind vectors, corresponding to different times and altitudes.

More formally: Consider a set of altitudes $H \subset [0, \infty)$ (in our data, these are 13 altitudes within the common flight range) and a set of times $T \subset [0, \infty)$ (usually spanning a period of 36h in 3h intervals). A *weather forecast* is a function $W : H \times T \to [0, \infty) \times [0, 2\pi)$. $W(h, \tau) = (r, \theta)$ encodes the wind speed r and the wind direction θ at altitude h and time τ. r is measured in $\frac{m}{s}$, and θ represents the angle between the segment and the wind vector.

Given an $h \in \mathbb{R}$ and/or a $\tau \in \mathbb{R}$ (i.e., not necessarily elements of H and T), we interpolate or extrapolate linearly in both dimensions in order to obtain the corresponding wind vector.

In order to be able to consider the effect of a wind vector on a segment w.r.t. fuel consumption and flight time, we need to introduce the concept of *air distance*, denoted by d^A. This is the distance traveled with respect to the air mass surrounding the aircraft. If there is no wind present (i.e., $r = 0$), the air distance is equal to the *ground distance* d^G, which is simply the great-circle-distance between the start and the end point. For $\theta = 0$ (tailwind), we have $d^A \le d^G$; whereas for $\theta = \pi$ (headwind), we have $d^A \ge d^G$. Since d^A is the distance traveled in the aircraft's frame of reference, it is the one that should be considered in aircraft performance calculations. More details follow in Section 1.1.1.

The basis for calculating the air distance is the following simple and intuitive relation:

$$\frac{d^A}{v^A} = \frac{d^G}{v^G}. \tag{1.1}$$

Here, v^A is the *airspeed*, which is an input parameter representing the speed relative to the surrounding air mass. v^G is the *ground speed*, i.e., the speed relative to the ground. This is calculated as follows:

$$v^G = \sqrt{(v^A)^2 - w_C^2} + w_T. \tag{1.2}$$

Here, w_C represents the *crosswind component* and w_T the *tailwind component* affecting the segment. These are the components of the wind vector with angles $\frac{\pi}{2}$ and 0 with respect to the segment's direction, respectively, see Figure 1.4. Since an aircraft's airspeed is always much larger than the wind speed, we can assume that v^G is always well-defined and positive.

By simple trigonometry, we have:

$$w_C = r\sin(\theta) \quad \text{and} \quad w_T = r\cos(\theta). \tag{1.3}$$

Combining equations (1.1), (1.2), and (1.3); we obtain

$$d^A = \frac{d^G v^A}{\sqrt{(v^A)^2 - r^2\sin(\theta)^2} + r\cos(\theta)}.$$

Since d^G and v^A are constant, we can thus calculate the air distance on a segment as a function of the speed and direction of the wind vector acting on it.

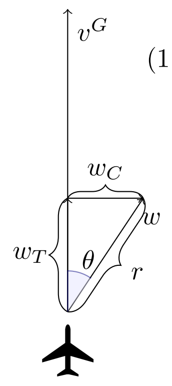
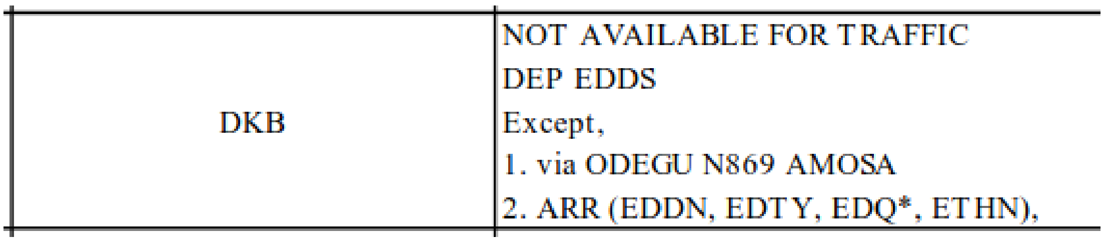


Figure 1.4: Crosswind and tailwind components.



DKB	NOT AVAILABLE FOR TRAFFIC DEP EDDS Except, 1. via ODEGU N869 AMOSA 2. ARR (EDDN, EDTY, EDQ*, ETHN),

Figure 1.5: Restriction ED3564, obtained from www.nm.eurocontrol.int/RAD/ on the 22nd of January 2023. Usage of waypoint DKB is not allowed in principle if the departure airport is EDDS. The only exceptions are if segment ODEGU→AMOSA on airway N869 is also included in the route, or if the arrival airport is one of a specified list.

1.1.4 Flight Restrictions

Besides the constraints described in Section 1.1.2, flight trajectories are heavily restricted, especially in high-traffic regions such as Europe or North America. The most prominent sources of such restrictions in the European airspace are *NOTAMs* and the *Route Availability Document (RAD)*.

NOTAMs (Notice(s) to Airmen, see [ICA23]) are restrictions issued by government agencies and airport operators, which limit the usage of certain parts of the Airway Network on short notice. They regulate the availability of waypoints, segments, runways, airspaces, etc., possibly only within given time or altitude ranges. Some common motivations for the implementation of these restrictions are military exercises, flights by heads of state, extreme weather phenomena or technical issues at an airport.

The main challenge of handling NOTAMs in a flight planning tool is that they might be published on very short notice. However, handling them in an optimization algorithm is straightforward, since the elements in question can simply be activated or deactivated prior to the search.

The Route Availability Document (see [EUR23]) is a collection of restrictions which are designed by European states and published by EUROCONTROL in a unified format. These restrictions are usually valid for a longer time period than NOTAMs and are significantly more complex. They are updated in 28-day cycles, but individual restrictions are often kept active for several years. Instead of limiting the usage of airway network elements in a straightforward manner like NOTAMs, they do so by making their availability dependent on convoluted conditions which are satisfied or not depending on the course of the trajectory. The main goal of these restrictions is to achieve a more efficient airspace usage by guiding the traffic flow a priori, so it conforms to the wishes of ATC authorities and fewer interventions from the controllers are needed.

In simple terms, RAD restrictions (which we also call *traffic restrictions*) can be understood as a set of logical expressions, all of which need to evaluate to true for feasibility. The literals in these expressions represent elements of the airway network, such as waypoints, segments, or airspaces. The evaluation of the expressions is done by setting all elements in the final trajectory to true, and all remaining ones to false.

See Figure 1.5 for a typical example. The departure and destination airports are known during the flight planning stage, but which waypoints or segments might be used is typically only determined during the optimization. Furthermore, the elements in a restriction can be arbitrarily far from each other. In our example, there are three segments separating DKB from

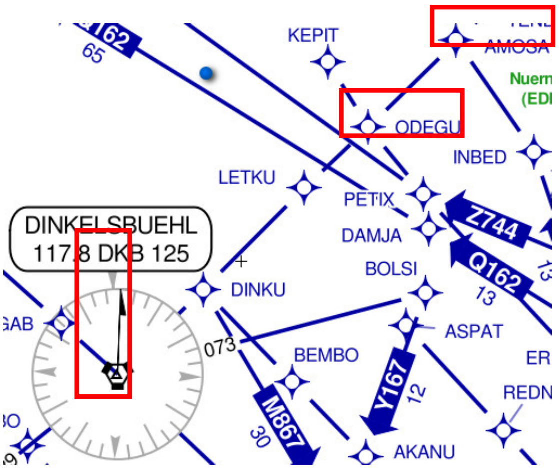


Figure 1.6: Airway network around waypoints DKB, ODEGU, and AMOSA. Screenshot obtained from skyvector.com on the 22nd of January 2023.

ODEGU→AMOSA, see Figure 1.6.

RAD restrictions make the flight planning problem particularly challenging. Not only do they make the optimization problem NP-hard, but their number has been steadily increasing over the past years. As of January 2023, the RAD document is a PDF with 679 pages, corresponding to over ten thousand constraints. In some cases, this makes it extremely challenging to find feasible solutions in an acceptable time, let alone optimal.

The motivation for the research in Chapter 6 was driven by the wish to understand the implications of restrictions in a theoretical sense, but the focus of this thesis is not on flight planning under consideration of restrictions. For more information on this topic, we refer to [KCL17] for an excellent overview and solution algorithm.

1.1.5 Objective Costs

Ultimately, the objective of the Flight Planning Problem is to minimize trajectory-dependent costs. We will distinguish three types of costs, which we present next.

Fuel and Time Costs

Fuel represents the biggest cost component in Flight Planning. To get an idea of the magnitude: The average price of jet fuel in the U.S. in 2019 and 2020 was 2.12 USD per gallon (0.7 USD/kg) and 1.97 USD per gallon (0.65 USD/kg), respectively [Adm21]. A typical consumption for a single flight from London to Singapore with an A380 is around 140t, meaning a fuel cost of over 90,000 USD. We assume a base fuel cost as part of the input, representing the price in USD/kg. Note that fuel consumption directly correlates with CO_2 emissions. Thus, it is possible to increase the base fuel cost to take these into account.

Time costs usually represent the sum of aircraft maintenance - and leasing costs, as well as crew salaries. Similar to the fuel costs, they are usually modeled via a base time cost given in USD/s. While not all the articles in this thesis explicitly consider time costs, they are trivial to incorporate in all models thanks to their linear nature.

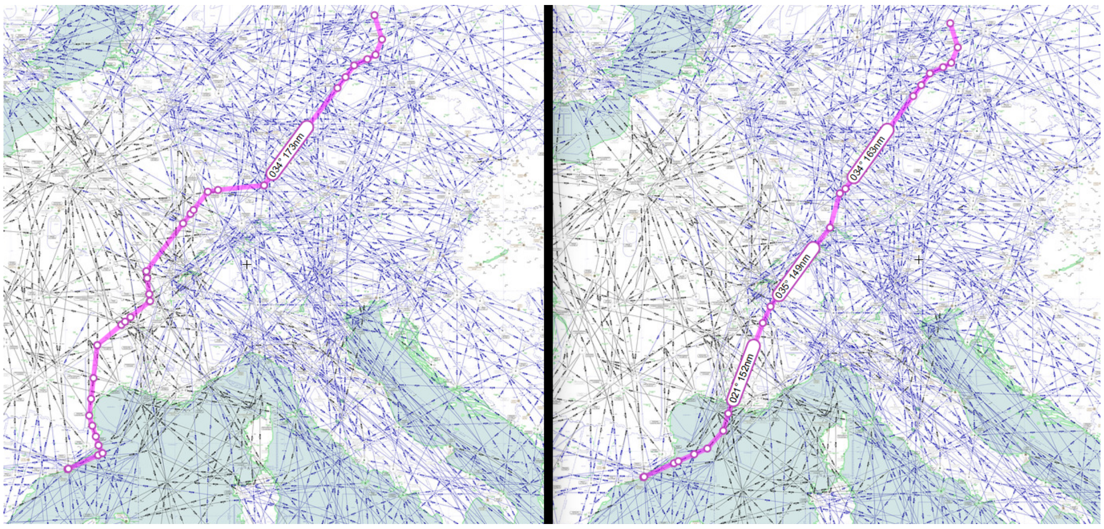


Figure 1.7: Image obtained using skyvector.com. On the left is a cost-minimal route from Barcelona to Berlin, considering all three cost components. On the right is the variant that only minimizes the sum of fuel and time. The route on the left takes a detour around the expensive Swiss airspace.

Overflight Costs

Overflight costs, also referred to as *overflight charges* or *overflight fees* are charged by countries to airlines for the right of crossing their airspace. While they are usually much smaller than fuel and time costs, they are significant enough to motivate airlines to deviate from the most fuel-efficient route in order to reduce the total costs. See Figure 1.7 for an example.

Depending on the cost stucture used by airspaces, the models that consider overflight charges range from trivial to NP-hard. The chapters in this thesis that consider overflight costs are 3 and 5. They both consider the *linear GCD* cost model and provide all the necessary background.

1.1.6 Importance of the Flight Planning Problem

From the previous sections, it should be evident that good flight planning algorithms are critical for the aviation industry. For airlines, these algorithms can lead to significant cost savings, as well as help them navigate the complex regulations and restrictions that are necessary for flight planning. By minimizing fuel consumption, which is typically the largest cost component, the environmental impact of flying is also reduced. This is an important measure that works in conjunction with other efforts such as the development of more efficient engines and biofuels.

However, it should be noted that more accurate solutions typically come at the cost of increased computational time. To be practical in production, flight planning algorithms need to be extremely fast, as routes need to be calculated a few hours before departure to take into account the most recent weather forecasts and restrictions. This makes the FPP a particularly challenging optimization problem, especially in sight of the NP-hardness of many of its components.

1.2 Literature Overview

While the study of several aviation topics such as pricing [Abd+19], crew scheduling [DÇ18], and many others [BS12] has a long tradition in the operations research community; the Flight Planning Problem has been mostly neglected in the mathematical optimization literature until the past few years.

It has, though, been partially studied in specialized literature, such as aviation journals. In very rough terms, it can be said that such works usually exhibit most of the following shortcomings, which make them inadequate for application in a modern, performance-critical flight planning system:

- Fuel and time consumption are computed by using analytical functions instead of aircraft performance tables, which are the standard in commercial aviation due to their accuracy.
- The objective function considers only fuel and/or time, ignoring overflight fees.
- Wind data is oversimplified (for example, ignoring time-dependency) or not considered at all.
- The horizontal search space is unrestricted.
- The optimization is either horizontal (i.e., assuming a constant flight altitude) or vertical (i.e., assuming a fixed horizontal routing).
- Experiments are done on small and/or artificially generated instances.

In the following, we present some of the most important contributions.

The work in [Jon74] is very thorough and application-driven. Apart from an optimal control algorithm, which is not feasible in practice due to the airway network and other constraints, it proposes a three-dimensional Dijkstra algorithm. This uses aircraft performance files and weather forecasts in a very similar form to today's. It also provides some computational results to demonstrate the feasibility of the approach. Unfortunately, nearly 50 years after the publication of this paper, the complexity and the size of the problem have grown dramatically. Nevertheless, the basic ideas presented by the author have proven their worth and are often used in modern approaches.

[HM98] uses a dynamic programming formulation to minimize fuel consumption during the cruise phase for a fixed horizontal route. Time constraints, for example requiring the aircraft to fly over a certain point during a given time interval, are handled by an a priori deactivation of the states in the search space which would violate this constraint. Furthermore, the authors use a neural network to speed-up the fuel consumption computations by a factor of up to 8.5.

[NSG14] computes a trajectory on a search space where the horizontal route is not restricted by waypoints and airways by splitting the problem into a horizontal and a vertical component. In the horizontal component, wind is taken into account, and a minimum-time route is computed by a dynamic programming approach based on integrating the dynamical equation for aircraft heading. For the vertical profile optimization, a theoretical model for aircraft performance data is used. The points at which a climb or descent phase may be initiated are defined a priori by dividing the horizontal route into equidistant waypoints. Finally, the vertical profile is obtained by solving another Dynamic Program. Experimental results are carried out using real-world weather forecast data. Two variants of the algorithm are compared, one that cruises on a single flight level, and one that allows cruise on multiple flight levels. Unsurprisingly, the latter yields better results. No information on running time of the algorithms is given.

[Kar+12a] gives a very realistic and detailed definition and overview of the FPP. The most relevant cost components and restrictions are presented, as well as an excellent review of previous work. The authors sketch some possible ways of solving the problem, such as a decomposition into horizontal and vertical optimization (2D+2D) or a 4D search. However, several of the complex components of the problem that the article describes are not considered in the algorithmic overview. In particular, the optimization methods described in the paper ignore aspects as essential as overflight fees and flight restrictions. Furthermore, the algorithms are presented at a very high level and no computational results are included.

A completely different approach is taken by [Yua+14], [Yua+15], [Mor+15] and [Mor+17]. These works focus on optimization of the vertical profile, by considering a simplified version of the problem. They rely on Second-Order Cone Programming and Non-Linear Programming, sometimes discretized in order to be able to model the problem as a MIP. Unfortunately, the reduced scope and the long runtimes make these approaches unsuitable for use in practice.

[Bon+13a] uses differential equations to describe the state of the aircraft, and proceeds to model the problem as a MINLP, which is solved using a branch-and-bound algorithm. It is one of the few works in the literature that considers overflight charges during the optimization. The computational results presented are obtained by assuming a constant flight altitude on a very small graph with 40 waypoints[b]. The runtime is close to 10 minutes for a single flight, which disqualifies this algorithm from practical use without major changes.

[BDW21], [BDW22], and [BDW23] focus on the *free-flight* variant of the problem, in which flight routes are not constrained by an airway network. They present impressive theoretical results, such as being able to find a globally optimal solution by combining discrete and continuous optimization. However, the assumptions used (no airway network, constant flight altitude, static weather) make these results more relevant for the theory than for practical use.

In recent years, a research group at the University of Southern Denmark has released a series of articles that represent the FPP in a very realistic way, and which present new algorithms tested on real-world data:

- [KCL16] focuses on optimizing vertical profiles given a horizontal route. It presents an A^* algorithm with a very good runtime. Although the problem does not satisfy the FIFO property due to the time-dependency of wind, the paper shows that algorithms relying on FIFO suffer from only a negligible loss of optimality, if at all.

- [KCL17] provides what is, to the best of our knowledge, the only algorithm in the literature designed to solve the Flight Planning Problem with RAD restrictions, for the case of constant altitude. The idea is to store multiple labels per node and to consider possible violations of restrictions during the label domination. The runtime is kept under control thanks to an iterative approach that gradually adds restrictions to be considered only as they are violated.

- [KCL18] presents a few variants of an A^* algorithm on a three-dimensional graph. It considers the problem almost in its full complexity (it does not consider overflight costs) and has very good computational results. Restrictions are handled using the multi-label technique from [KCL17]. We discuss this paper in detail in Chapter 4.

- [JCL17b] focuses on free-flight optimization. Here, the flight route is not subject to the Airway Network, but is allowed to choose an arbitrary path. While this problem might play a bigger role in the future due to changing regulations, it is not relevant for this thesis.

[b]For reference, the data we use for our experiments contains close to 100,000 waypoints.

Finally, in [Bla+16e] the authors (including the author of this thesis) present a comprehensive overview of the complexity of flight planning with overflight costs, as well as algorithms to solve some of the variants.

1.3 Outline of this Thesis

This dissertation is a compilation of publications produced as part of my PhD studies. Chapters 2-6 each contain a literal transcription of a research article. The previously unpublished content consists of this general introduction (Chapter 1), the individual introductions for Chapters 2-6, the appendices 5.8 and 6.4, and the general conclusion (Chapter 7). It is worth noting that the chapters in this thesis are not arranged in chronological order. Instead, they are organized in a way that highlights and groups the key themes and concepts of the research, in order to facilitate the reading.

Chapter 2 was published in [Bla+16d] and received the ATMOS 2016 Best Paper Award. In it, we investigate a variant of the Flight Planning Problem that aims to minimize flight time under consideration of weather, while restricted to a constant flight altitude. We introduce a fundamental technique for reliably underestimating flight time on individual segments, which allows us to develop a very efficient A^* algorithm.

Chapter 3 was published in [SMB19]. It introduces *pruning* techniques to reduce the size of the graph before starting the search. The pruning is based on underestimating fuel, time, and overflight costs for each node in the graph. For each node, the cost underestimator is less than or equal to the cost of any route from departure to destination via that node. By comparing to an upper bound, obtained from a feasible solution, we are able to exclude large parts of the graph from the search without endangering optimality. In other words, the search space reduction is done a priori instead of on-the-fly as with A^* algorithms. We investigate the effect of this reduction on search algorithms.

Chapter 4 was published in [BBC22]. As in Chapter 2, it presents an A^* algorithm that relies on Super Optimal Wind. It extends the previous results by optimizing on the 3D space and not just vertically. We use the concept and properties of *Idealized Vertical Profiles* to calculate the potential function needed for A^*.

Chapter 5 was published in [Bla+17b]. It examines the *Shortest Path Problem with Crossing Costs* (SPPCC), which is a simplified version of the Flight Planning Problem at constant altitude under consideration of overflight fees. It focuses on the cost model most prevalent in Europe, in which each country calculates overflight fees for a flight based on the great-circle-distance between entry point and exit point. This cost structure breaks the subpath optimality property that makes Dijkstra algorithms optimal for the classical shortest-path problem. The main contribution of the chapter is the *cost projection* technique, which constructs artificial costs on the arcs in such a way that the estimated total cost is as close as possible to the real total cost. This chapter also introduces the *Two-Layer Dijkstra* algorithm, which is the only known method in the literature for solving the SPPCC to optimality in polynomial time.

Finally, Chapter 6 was published in [Bla+15]. The focus is the *Path Avoiding Forbidden Pairs (PAFP)* Problem. This is a subvariant of the Flight Planning Problem under consideration of RAD restrictions as described in Section 1.1.4. The goal of the PAFP problem is to find a path on a graph between two given nodes in such a way that, for each pair of nodes given in the input, the path contains at most one of them. Restrictions of this type are very common among RAD restrictions, and the problem is known to be NP-hard. In this chapter, we present the first polyhedral study of the PAFP polytope.

1.3.1 Individual Contributions

As part of the requirements for this thesis, here I provide a clear overview of my personal contributions to each of the research articles that are included as main chapters of the dissertation. This information is presented in a structured format according to the type of contribution I made for each paper.

Research Question

I played a primary role in the initial conception of the research questions that led to all papers.

For [Bla+16d], I had the original idea of investigating wind-dependent underestimators after realizing that interpolating air distance does not produce valid lower bounds.

The central ideas in [SMB19] were also mine: A priori pruning of nodes based on lower and upper bounds, using tank capacity as a fallback option, using a macro graph to underestimate the overflight costs, and using optimal vertical profiles in conjunction with Super-Optimal Wind for underestimating the fuel costs.

In [Bla+17b] I had the idea for the Two-Layer Dijkstra algorithm and for creating artificial arc costs by solving a linear program.

The main idea behind [BBC22] of creating an A^* algorithm for the three-dimensional case, was mine.

So was the idea of looking for facets of the PAFP polytope in [Bla+15].

Literature Research

I conducted most of the literature research needed for the papers. A large part of it was derived from the extensive research on flight planning that I conducted over the course of several years for the research project that led to the creation of this thesis.

Additionally, most of the works relevant for specific papers were found by me. Some examples of this are the articles on Contraction-Hierarchies (used in [Bla+16d]), A^* in flight planning (cited in [BBC22]) and the Path Avoiding Forbidden Pairs Problem (essential for [Bla+15]).

Algorithm Design and Theoretical Results

The mathematical models in all papers were defined mostly by me.

In [Bla+16d], I actively collaborated with coauthors on the theoretical part. There, I contributed both with high-level (e.g. how to define the Super-Optimal Wind) and low-level (e.g. the mathematical proofs) ideas. I had the idea of comparing our algorithm against Time-Dependent Contraction Hierarchies.

The central ideas in [SMB19] were also mine: A priori pruning of nodes based on lower and upper bounds, using tank capacity as a fallback option, using a macro graph to underestimate the overflight costs, and using optimal vertical profiles in conjunction with Super-Optimal Wind for underestimating the fuel costs.

In [Bla+17b], I played a very active role in the iterations that led to the final LP formulation, as well as in the definition and construction of *good paths*.

As for [BBC22], apart from some valuable brainstorming sessions with my coauthors to define a good potential function, all ideas and proofs were mine.

For [Bla+15], I coordinated and supervised the computational calculation of the polyhedral description for specific examples. After many discussions with coauthors and analysis of these examples, I found the correct formulation of the inequalities that would later prove to be facet-defining. I also came up with all the proofs entirely on my own.

Implementation

The implementation of the algorithms in [Bla+16d], [SMB19], and [Bla+17b] were built as extensions of a flight planning prototype developed at the Zuse Institute Berlin, of which I was the main developer. This significantly reduced the implementation effort for all these works. As for the additional work specific to these papers, I contributed mostly in the form of supervision, advice, and debugging assistance.

[BBC22] was implemented separately from this prototype. This work was entirely done by me.

Note that [Bla+15] has no computational component.

Computational Experiments

For [Bla+16d], [SMB19], and [Bla+17b], I played a primary role in the design and supervision of all computational experiments and the presentation of their results. Additionally, in [Bla+16d], I was the main responsible for setting up and running the tests of the Contraction Hierarchies and Time-Dependent Contraction Hierarchies algorithms that we used as benchmarks.

For [BBC22], I did a large majority of the work collecting the data for the experiments. I designed, ran, and evaluated the tests on my own.

Manuscript Preparation

In the case of [SMB19], my involvement was limited to discussing the high-level structure of the document and proofreading. I also participated in discussing the issues raised by the reviewers and finding solutions.

I played a primary role in the structuring and writing of the other four papers. I was also the corresponding author and did a majority of the corrections in the review process.

References

[22a] *IATA Fuel Fact Sheet.* [Online; accessed 15-Jan-2023]. 2022. URL: `https://www.iata.org/en/iata-repository/pressroom/fact-sheets/fact-sheet---fuel/`.

[22b] *WAWFOR (World Aviation Weather FORecast).* [Online; accessed 18-April-2022]. 2022. URL: `https://www.dwd.de/EN/ourservices/aviation_wawfor/wawfor_node.html`.

[22c] *World Area Forecast System.* [Online; accessed 18-April-2022]. 2022. URL: `https://aviationweather.gov/wafs`.

[Abd+19] Juhar Ahmed Abdella, Nazar Zaki, Khaled Shuaib, and Fahad Khan. “Airline ticket price and demand prediction: A survey”. In: *Journal of King Saud University - Computer and Information Sciences* (2019). ISSN: 1319-1578. DOI: `https://doi.org/10.1016/j.jksuci.2019.02.001`. URL: `https://www.sciencedirect.com/science/article/pii/S131915781830884X`.

[Adm21] U.S. Energy Information Administration. *U.S. Kerosene-Type Jet Fuel Retail Sales by Refiners.* [Online; accessed 5-May-2021]. 2021. URL: `https://www.eia.gov/dnav/pet/hist/LeafHandler.ashx?n=pet&s=ema_epjk_ptg_nus_dpg&f=a`.

[BBC22] Marco Blanco, Ralf Borndörfer, and Pedro Maristany de las Casas. “An A* Algorithm for Flight Planning Based on Idealized Vertical Profiles”. In: *22nd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems.* Vol. 106. This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit `http://creativecommons.org/licenses/by/4.0/`. 2022, 1:1–1:15. DOI: `https://doi.org/10.4230/OASIcs.ATMOS.2022.1`.

[BDW21] Ralf Borndörfer, Fabian Danecker, and Martin Weiser. “A Discrete-Continuous Algorithm for Free Flight Planning”. In: *Algorithms* 14.1 (2021), p. 4. DOI: `10.3390/a14010004`.

[BDW22] Ralf Borndörfer, Fabian Danecker, and Martin Weiser. “A Discrete-Continuous Algorithm for Globally Optimal Free Flight Trajectory Optimization”. In: *22nd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2022).* Vol. 106. 2022, pp. 1–13. DOI: `10.4230/OASIcs.ATMOS.2022.2`.

[BDW23] Ralf Borndörfer, Fabian Danecker, and Martin Weiser. *Newton’s Method for Global Free Flight Trajectory Optimization.* eng. Tech. rep. 23-08. Takustr. 7, 14195 Berlin: ZIB, 2023.

[Bla+15] Marco Blanco, Ralf Borndörfer, Michael Brückner, Nam Dũng Hoàng, and Thomas Schlechte. “On the Path Avoiding Forbidden Pairs Polytope”. In: *Electronic Notes in Discrete Mathematics* 50 (2015). LAGOS’15 - VIII Latin-American Algorithms, Graphs and Optimization Symposium. This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License. To view a copy of this license, visit `https://creativecommons.org/licenses/by-nc-nd/4.0/`., pp. 343–348. ISSN: 1571-0653. DOI: `10.1016/j.endm.2015.07.057`.

[Bla+16d] Marco Blanco, Ralf Borndörfer, Nam Dũng Hoàng, Anton Kaier, Adam Schienle, Thomas Schlechte, and Swen Schlobach. “Solving Time Dependent Shortest Path Problems on Airway Networks Using Super-Optimal Wind”. In: *16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2016).* Vol. 54. This work is licensed under the Creative Commons Attribution 3.0 Unported License. To view a copy of this license, visit `http://creativecommons.org/licenses/by/3.0/`. 2016. DOI: `10.4230/OASIcs.ATMOS.2016.12`.

[Bla+16e] Marco Blanco, Ralf Borndörfer, Nam Dũng Hoàng, Anton Kaier, Thomas Schlechte, and Swen Schlobach. *The Shortest Path Problem with Crossing Costs.* eng. Tech. rep. Berlin, 2016.

[Bla+17b] Marco Blanco, Ralf Borndörfer, Nam Dung Hoàng, Anton Kaier, Pedro M. Casas, Thomas Schlechte, and Swen Schlobach. “Cost Projection Methods for the Shortest Path Problem with Crossing Costs”. In: *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017).* Ed. by Gianlorenzo D’Angelo and Twan Dollevoet. Vol. 59. OpenAccess Series in Informatics (OASIcs). This work is licensed under the Creative Commons Attribution 3.0 Unported License. To view a copy of this license, visit `http://creativecommons.org/licenses/by/3.0/`. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, 15:1–15:14. ISBN: 978-3-95977-042-2. DOI:

10.4230/OASIcs.ATMOS.2017.15. URL: http://drops.dagstuhl.de/opus/volltexte/2017/7893.

[Bon+13a] Pierre Bonami, Alberto Olivares, Manuel Soler, and Ernesto Staffetti. "Multiphase Mixed-Integer Optimal Control Approach to Aircraft Trajectory Optimisation". In: *Journal of Guidance, Control, and Dynamics* (2013), pp. 1267–1277.

[BS12] Cynthia Barnhart and Barry Smith, eds. *Quantitative Problem Solving Methods in the Airline Industry.* International Series in Operations Research and Management Science 978-1-4614-1608-1. Springer, 2012. DOI: 10.1007/978-1-4614-1608-1. URL: https://ideas.repec.org/b/spr/isorms/978-1-4614-1608-1.html.

[DÇ18] Muhammet Deveci and Nihan Çetin Demirel. "A survey of the literature on airline crew scheduling". In: *Engineering Applications of Artificial Intelligence* 74 (June 2018). DOI: 10.1016/j.engappai.2018.05.008.

[Eur22] European Environment Agency. *European Aviation Environmental Report 2022 - Executive Summary.* Online, https://www.easa.europa.eu/eco/sites/default/files/2022-09/EnvironmentalReport_EASA_summary_12.pdf. Accessed: 31 October 2022. 2022.

[EUR23] EUROCONTROL. *Route Availability Document.* [Online; accessed 10-March-2023]. 2023. URL: https://www.nm.eurocontrol.int/RAD/.

[HM98] Patrick Hagelauer and Felix Antonio Claudio Mora-Camino. "A soft dynamic programming approach for on-line aircraft 4D-trajectory optimization". In: *European Journal of Operational Research* 107.1 (May 1998), pp. 87–95. DOI: 10.1016/S0377-2217(97)00221-X. URL: https://hal-enac.archives-ouvertes.fr/hal-01021633.

[ICA23] ICAO. *ICAO Safety - NOTAMs.* [Online; accessed 10-March-2023]. 2023. URL: https://www.icao.int/safety/istars/pages/notams.aspx.

[Int22] International Air Transport Association. *Air Passenger Numbers to Recover in 2024.* Online, https://www.iata.org/en/pressroom/2022-releases/2022-03-01-01/. Accessed: 31 October 2022. 2022.

[JCL17b] Casper Kehlet Jensen, Marco Chiarandini, and Kim S. Larsen. "Flight Planning in Free Route Airspaces". In: *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017).* Ed. by Gianlorenzo D'Angelo and Twan Dollevoet. Vol. 59. OpenAccess Series in Informatics (OASIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, pp. 1–14. ISBN: 978-3-95977-042-2. DOI: 10.4230/OASIcs.ATMOS.2017.14.

[Jon74] H.M. de Jong. *Optimal Track Selection and 3-dimensional flight planning.* Tech. rep. 1974.

[Kar+12a] Stefan E. Karisch, Stephen S. Altus, Goran Stojković, and Mirela Stojković. "Operations". English. In: *Quantitative Problem Solving Methods in the Airline Industry.* Ed. by Cynthia Barnhart and Barry Smith. Vol. 169. International Series in Operations Research & Management Science. Springer US, 2012, pp. 283–383. ISBN: 978-1-4614-1607-4.

[KCL16] Anders Nicolai Knudsen, Marco Chiarandini, and Kim S. Larsen. "Vertical Optimization of Resource Dependent Flight Paths". In: *ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016).* 2016, pp. 639–645. DOI: 10.3233/978-1-61499-672-9-639.

[KCL17] Anders Nicolai Knudsen, Marco Chiarandini, and Kim S. Larsen. "Constraint Handling in Flight Planning". In: *Principles and Practice of Constraint Programming - 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings.* 2017, pp. 354–369. DOI: 10.1007/978-3-319-66158-2_23. URL: https://doi.org/10.1007/978-3-319-66158-2_23.

[KCL18] Anders Nicolai Knudsen, Marco Chiarandini, and Kim S. Larsen. "Heuristic Variants of A* Search for 3D Flight Planning". In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 15th International Conference, CPAIOR 2018, Delft, The Netherlands, June 26-29, 2018, Proceedings.* 2018, pp. 361–376. DOI: 10.1007/978-3-319-93031-2_26. URL: https://doi.org/10.1007/978-3-319-93031-2_26.

[Mor+15] Liana Amaya Moreno, Zhi Yuan, Armin Fügenschuh, Anton Kaier, and Swen Schlobach. "Combining NLP and MILP in Vertical Flight Planning". In: *Operations Research Proceedings 2015, Selected Papers of the International Conference of the German, Austrian and Swiss Operations Research Societies (GOR, ÖGOR, SVOR/ASRO), University of Vienna, Austria, September 1-4, 2015.* 2015, pp. 273–278. DOI: 10.1007/978-3-319-42902-1_37. URL: https://doi.org/10.1007/978-3-319-42902-1_37.

[Mor+17] Liana Amaya Moreno, Armin Fügenschuh, Anton Kaier, and Swen Schlobach. "A Nonlinear Model for Vertical Free-Flight Trajectory Planning". In: *Operations Research Proceedings 2017, Selected Papers of the Annual International Conference of the German Operations Research Society (GOR), Freie Universiät Berlin, Germany, September 6-8, 2017.* 2017, pp. 445–450. DOI: 10.1007/978-3-319-89920-6_59. URL: https://doi.org/10.1007/978-3-319-89920-6_59.

[NSG14] Hok K. Ng, Banavar Sridhar, and Shon Grabbe. "Optimizing Aircraft Trajectories with Multiple Cruise Altitudes in the Presence of Winds". In: *Journal of Aerospace Information Systems* 11.1 (2014), pp. 35–47.

[Sil10] Saulo Silva. *A Brief History of RVSM.* [Online; accessed 10-December-2020]. 2010. URL: https://www.icao.int/esaf/documents/rvsm/2010/afi%20riss/presentations.pdf.

[SMB19] Adam Schienle, Pedro Maristany, and Marco Blanco. "A Priori Search Space Pruning in the Flight Planning Problem". In: *19th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2019).* Ed. by Valentina Cacchiani and Alberto Marchetti-Spaccamela. Vol. 75. OpenAccess Series in Informatics (OASIcs). This work is licensed under the Creative Commons Attribution 3.0 Unported License. To view a copy of this license, visit http://creativecommons.org/licenses/by/3.0/. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, 8:1–8:14. ISBN: 978-3-95977-128-3. DOI: 10.4230/OASIcs.ATMOS.2019.8.

[Yua+14] Zhi Yuan, Armin Fügenschuh, Anton Kaier, and Swen Schlobach. "Variable Speed in Vertical Flight Planning". In: *Operations Research Proceedings 2014, Selected Papers of the Annual International Conference of the German Operations Research Society (GOR), RWTH Aachen University, Germany, September 2-5, 2014.* 2014, pp. 635–641. DOI: 10.1007/978-3-319-28697-6_88. URL: https://doi.org/10.1007/978-3-319-28697-6_88.

[Yua+15] Zhi Yuan, Liana Amaya Moreno, Armin Fügenschuh, Anton Kaier, and Swen Schlobach. "Discrete Speed in Vertical Flight Planning". In: *Computational Logistics - 6th International Conference, ICCL 2015 Delft, The Netherlands, September 23-25, 2015. Proceedings.* 2015, pp. 734–749. DOI: `10.1007/978-3-319-24264-4\_50`. URL: `https://doi.org/10.1007/978-3-319-24264-4_50`.

1.4 Appendix: Aircraft Performance and Propagation

In this section, we will introduce aircraft performance functions and their usage at increasing levels of complexity. First, we will describe the data provided by the aircraft manufacturer. Then, we will see how to use that data for the calculation of basic flight phases. Finally, in Table 1.3, we present the propagation functions used in more complex flight procedures, which are the ones we will use in our algorithms.

Besides fuel consumption, some factors of interest are climb/descent angles, and the time needed to complete the current phase.

Next, we introduce some basic aeronautical definitions.

Definition 1.5. A *flight mode* describes the vertical movement of an aircraft, and is one of the following three:

1. *Climb*: The aircraft increases its altitude.
2. *Cruise*: The aircraft flies at constant altitude.
3. *Descent*: The aircraft decreases its altitude.

In practice, it is necessary to consider *take-off* and *landing* modes as well in order to accurately model the vertical profile. However, from the algorithmic point of view, the corresponding performance functions are simplified versions of those for climb and descent, respectively. For the sake of simplicity and without loss of generality, we ignore them in this work.

Definition 1.6. A *flight phase* is a maximal period of time during which the aircraft flies in a single mode.

Note that a flight may encompass multiple phases of the same mode and usually does, see Figure 1.8 for a simple example.

Finally, we define an aircraft performance function for each mode.

The input and output parameters of aircraft performance functions vary for the different modes. For instance, the only output value of interest during a cruise phase is fuel consumption, while in a climb phase it is also necessary to compute the climb angle [c]. Similarly, by definition, the climb- and descend functions need a target altitude, which for cruise is the same as the initial altitude. The following is a complete list of parameters which are used as input or output in aircraft performance functions.

- w_I: Initial weight.
- h_I: Initial altitude.
- h_T: Target altitude.
- Δ_t: Time used to complete the phase.
- Δ_f: Fuel consumed during the phase.
- Δ_d: Horizontal distance flown during the phase.
- Δ_h: Change of altitude during the phase.

[c]It is possible to influence the climb angle by varying the flight speed, but in this work we restrict ourselves to the case of constant speed.

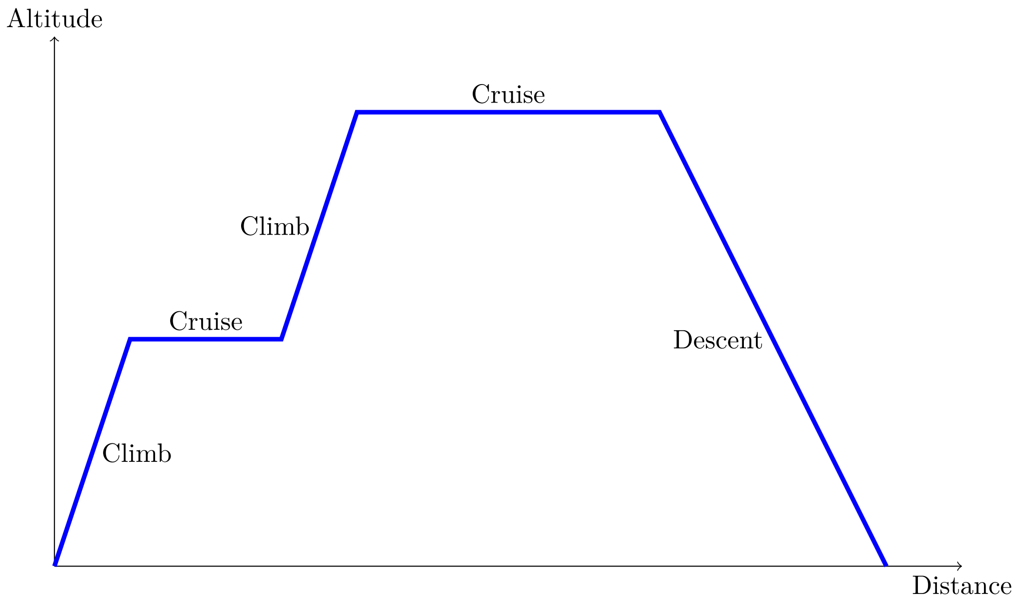


Figure 1.8: A vertical flight profile with two climb phases, two cruise phases, and one descent phase.

The possible range of the weight w_I is the interval $[w_{DOW}, w_{MTOW}] \subset \mathbb{R}_+$, which is defined by the aircraft's dry operating weight and maximum take-off weight. The altitudes that need to be considered also vary depending on the aircraft, but we can safely say that they lie between 386m below sea level (the lowest altitude of an airfield) and 18288m above sea level (the altitude reached by the Concorde, far higher than anything reachable by the aircraft in our data set).

Even for input points contained in the tables, the output is sometimes not defined. This is most common in the climb tables for combinations of a high weight and a high target altitude. In these cases, a direct climb is simply not possible for the engines. This is the reason why vertical flight profiles typically follow a "staircase" pattern between take-off and the reaching of the final cruise altitude: Cruising for some time at a suboptimal altitude burns enough fuel to reach a weight that allows further climbing. See Figure 1.9 for a real life example.

In Table 1.1, we present an overview of the input and output parameters corresponding to the different flight phases. Note that Δ_d is both an input parameter for the cruise function and an output parameter for the climb and descent functions.

Table 1.1: Input and output parameters of the various aircraft performance functions

Flight phase	**Input**	**Output**
Climb	w_I, h_I, h_T	$\Delta_f, \Delta_t, \Delta_d$
Cruise	w_I, h_I, Δ_d	Δ_f
Descent	w_I, h_I, h_T	$\Delta_f, \Delta_t, \Delta_d$

These functions are not given by explicit formulas. In day-to-day flight planning, aircraft

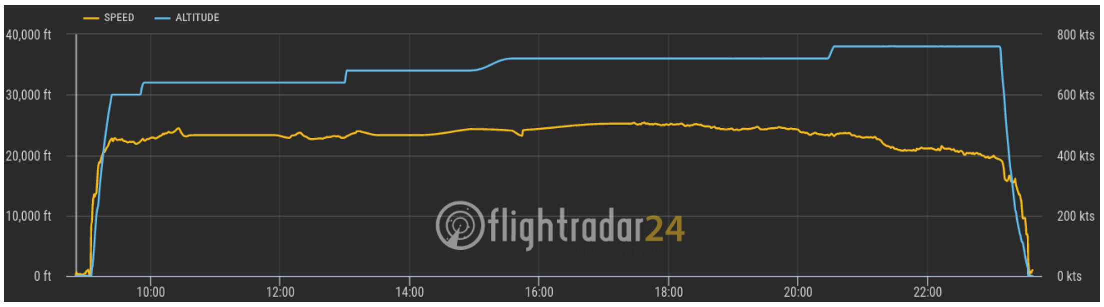


Figure 1.9: Image obtained from FlightRadar24.com on the 9th of July 2021. The blue graph represents the aircraft's altitude over time.

performance functions are provided by manufacturers in the form of look-up tables. The values read from the tables are interpolated to obtain the desired precision. A few (somewhat outdated) examples of such tables can be found in [Jon74].

While it is tempting to approximate the functions given from the tables with analytical functions, as it could for instance allow us to derive theoretical results such as approximation guarantees, we have not been able to find approximations that yield an acceptable error. The look-up tables for big aircraft contain tens of thousands of data points, and a deviation by even a few kg at a given point can lead to a completely different solution. This issue is confirmed in [HM98]. For that reason, in the remainder of this work we will treat aircraft performance functions as look-up tables.

To give the reader an idea of what these tables look like, we can make the following statements, which are quite intuitive:

- As the aircraft's weight decreases, so does fuel consumption. This means that, for example, w_I and Δ_f are proportional for all three flight phases.
- Similarly, a lighter aircraft weight leads to a steeper climb/descent angle, meaning that as w_I increases, Δ_t and Δ_d also increase.
- Each aircraft has an *optimum cruise altitude*. Below this altitude, higher h_I and h_T lead to a lower Δ_f in all functions. Above this altitude, the opposite is the case. [d]
- In general, climbing over a given distance burns more fuel than cruising the same distance, which is in turn less efficient than descending over that distance. Here, we assume the same starting weight in all three phases and roughly similar altitudes.
- Some entries of the tables are undefined, in particular the climb tables for high values of w_I and h_T.

These properties are essential for a good algorithm design. For example, [KCL18] makes use of the optimum cruise altitude in a label domination heuristic; and bases the feasibility of a

[d]While the aerodynamics behind this phenomenon are complex, a simple explanation is that air is less dense at higher altitudes, which results in less drag on the aircraft, allowing it to fly more efficiently. Additionally, the engines of a jet aircraft are designed to operate most efficiently at higher altitudes, where the air is cooler and has a higher oxygen content. On the other hand, at extremely high altitudes, the air becomes so thin that the engines struggle to produce enough thrust to maintain the aircraft's speed and altitude. Additionally, the air at these altitudes is so cold that it can cause damage to the engines and other systems on the aircraft.

potential function used for an A* algorithm on the fact that descent is the most efficient phase. We will also use this property in Chapter 4.

For inputs that don't match a given data point, we use linear interpolation/extrapolation, just like we do for weather data.

Sometimes, it is desired to climb or descend continuously between two given coordinates, as we will see in the following sections. That is, we are given a target distance instead of a target altitude, and the change in altitudes is an output. This can be achieved by identifying a target altitude that is represented in the tables and leads to a longer distance, and then scaling all output parameters. The same thing applies to the source altitude. This is possible because the climb and descend functions are usually very close to linear.

Next, we illustrate the interplay between the wind data presented in Section 1.1.3 and aircraft performance calculations.

- For cruise on a segment at a given altitude and a given time, we define the input distance Δ_d as the air distance calculated with that segment and altitude, at the time of entering the segment. The cruise time is not an output of the performance tables, but can be easily calculated as $\Delta_t = \frac{d^A}{v^A}$.
- For climb and descent to a given altitude, we define the input distance Δ_d as the air distance on the segment at the time of entering it, and at altitude $\frac{h_I + h_T}{2}$.
- For climb and descent over a given distance, we run the calculation without weather and then scale all output values by the ratio between the length of the segment and the air distance at the average altitude, as just described. [e]

Using all the information provided in this Section, we can define a set of functions which calculate the aircraft performance data under consideration of weather:

Table 1.2: Base propagation functions

Flight phase	**Function name**	**Input**	**Output**
Cruise	ξ_{CR}	w_I, h_I, Δ_d, t_I	Δ_f, Δ_t
Climb a given distance	ξ_{CD}	w_I, h_I, Δ_d, t_I	$\Delta_f, \Delta_t, \Delta_h$
Climb to a given altitude	ξ_{CH}	w_I, h_I, h_T, t_I	$\Delta_f, \Delta_t, \Delta_d$
Descend a given distance	ξ_{DD}	w_I, h_I, Δ_d, t_I	$\Delta_f, \Delta_t, \Delta_h$
Descend to a given altitude	ξ_{DH}	w_I, h_I, h_T, t_I	$\Delta_f, \Delta_t, \Delta_d$

Here, we follow the previous notation and use t_I to denote the current time, which is needed for the weather calculations.

As mentioned before, the functions in Table 1.2 are not defined for all inputs. When that is the case, we assume that each entry of the output vector is ∞.

In practice, a climb or descent to a target altitude is always followed by a cruise (see Section 1.1.2). In these cases, we get as input both a fixed distance and a fixed altitude. The aircraft climbs or descends until reaching the target altitude, and then cruises the remaining distance. This procedure is called *step climb* and *step descent*, respectively.

For example, given w_I, h_I, Δ_d, t_I, and h_T as input, with $h_I < h_T$, we first compute $(f, t, d) := \xi_{CH}(w_I, h_I, \Delta_d, t_I)$. If the climb distance d is larger than the input distance Δ_d, we abort, since

[e]The altitude we consider is not the average of the initial and final altitudes, as the latter is still unknown. In its stead, we consider the "fake" target altitude described before.

a step climb is not possible within that distance. Otherwise, we compute $\xi_{CR}(w_I - w, h_T, \Delta_d - d, t_I + t)$ to obtain our final result. We define the resulting composed function as $\xi_{SC} : \mathbb{R}^5 \to \mathbb{R}^2$, with $(w_I, h_I, \Delta_d, t_I, h_T) \mapsto (\Delta_f, \Delta_t)$. For step descent, we define ξ_{SC} analogously.

In Table 1.3, we can finally present the propagation functions that will be used in the remainder of this thesis. The new functions are just copies of the ones introduced before (see column "Base function" in the table), with some additional trivial output. We introduce them to simplify notation and standardize input and output. Note that now the output type of all functions is the same. We fill the missing gaps trivially: The change of altitude is zero for cruise and $h_T - h_I$ for step climb and step descent.

Flight procedure	**Function name**	**Base function**	**Input**	**Output**
Step climb	ρ_{SC}	ξ_{SC}	$w_I, h_I, \Delta_d, t_I, h_T$	$\Delta_f, \Delta_t, \Delta_h$
Cruise	ρ_{CR}	ξ_{CR}	w_I, h_I, Δ_d, t_I	$\Delta_f, \Delta_t, \Delta_h$
Transition climb	ρ_{TC}	ξ_{CD}	w_I, h_I, Δ_d, t_I	$\Delta_f, \Delta_t, \Delta_h$
Step descent	ρ_{SD}	ξ_{DH}	$w_I, h_I, \Delta_d, t_I, h_T$	$\Delta_f, \Delta_t, \Delta_h$
Transition descent	ρ_{TD}	ξ_{DD}	w_I, h_I, Δ_d, t_I	$\Delta_f, \Delta_t, \Delta_h$

Table 1.3: Final propagation functions

These functions are then combined into the function ρ used in Chapter 4.

Introduction to Chapter 2

This chapter was published in [Bla+16d]. That article was the recipient of the Best Paper Award at the 16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2016). It considers a simplified version of the Flight Planning Problem that aims to minimize travel time while maintaining a constant altitude.

There are several reasons why this variant of the problem is significant: One is its integration in hybrid algorithms that optimize the horizontal route and the vertical profile separately. Additionally, as we argue in the chapter, cruise is the longest and most crucial phase of long-haul flights. Fuel consumption is proportional to travel time during the cruise phase, thus making the objective function more general than it seems. By limiting the graph to a constant altitude, we also have the opportunity to compare our algorithm against Time-Dependent Contraction Hierarchies [Bat+13], which takes several hours to preprocess on our instances and would not be feasible with a three-dimensional graph.

Most importantly, the central techniques developed in this chapter are easy to integrate in more general settings. In fact, in Chapters 3 and 4 we rely on such underestimation techniques.

The findings of this research not only have implications for Flight Planning, but also reveal a significant limitation of algorithms such as (Time-Dependent) Contraction Hierarchies, which are commonly regarded as the gold standard for shortest path calculations. We have shown that these algorithms might struggle when applied to instance families other than those for which they were originally designed (road networks), highlighting the need for alternative approaches depending on the application.

For the interested reader, an extended version of this chapter can be found in [Sch16a]. This includes, for example, a sufficient condition on the wind forecast for the FIFO property to be satisfied.

References

[Bat+13] G. Veit Batz, Robert Geisberger, Peter Sanders, and Christian Vetter. "Minimum Time-dependent Travel Times with Contraction Hierarchies". In: *Journal of Experimental Algorithmics* 18 (Apr. 2013), 14:11–14:143. ISSN: 1084-6654.

[Bla+16d] Marco Blanco, Ralf Borndörfer, Nam Dũng Hoàng, Anton Kaier, Adam Schienle, Thomas Schlechte, and Swen Schlobach. "Solving Time Dependent Shortest Path Problems on Airway Networks Using Super-Optimal Wind". In: *16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2016)*. Vol. 54. This work is licensed under the Creative Commons Attribution 3.0 Unported License. To view a copy of this license, visit `http://creativecommons.org/licenses/by/3.0/`. 2016. DOI: `10.4230/OASIcs.ATMOS.2016.12`.

[Sch16a] Adam Schienle. “Shortest Paths on Airway Networks”. MA thesis. Freie Universität Berlin, 2016, p. 66.

Chapter 2

Solving Time Dependent Shortest Path Problems on Airway Networks Using Super-Optimal Wind

Marco Blanco, Ralf Borndörfer, Nam-Dũng Hoang, Anton Kaier, Adam Schienle, Thomas Schlechte, Swen Schlobach

Abstract

We study the Flight Planning Problem for a single aircraft, which deals with finding a path of minimal travel time in an airway network. Flight time along arcs is affected by wind speed and direction, which are functions of time. We consider three variants of the problem, which can be modeled as, respectively, a classical shortest path problem in a metric space, a time-dependent shortest path problem with piecewise linear travel time functions, and a time-dependent shortest path problem with piecewise differentiable travel time functions.

The shortest path problem and its time-dependent variant have been extensively studied, in particular, for road networks. Airway networks, however, have different characteristics: the average node degree is higher and shortest paths usually have only few arcs.

We propose A* algorithms for each of the problem variants. In particular, for the third problem, we introduce an application-specific “super-optimal wind” potential function that overestimates optimal wind conditions on each arc, and establish a linear error bound. We compare the performance of our methods with the standard Dijkstra algorithm and the Contraction Hierarchies (CHs) algorithm. Our computational results on real world instances show that CHs do not perform as well as on road networks. On the other hand, A* guided by our potentials yields very good results. In particular, for the case of piecewise linear travel time functions, we achieve query times about 15 times shorter than CHs.

2.1 Introduction

We consider the *Flight Planning Problem* (FPP), which seeks to compute a cost-minimal flight trajectory on an airway network, given origin and destination airports, a departure time, an aircraft, and weather prognoses. Some of the factors that need to be taken into account are overflight costs, weight-dependent fuel consumption functions, avoidance of hazardous areas, and restrictions to prevent overcrowding of airspaces, such as those published by EUROCONTROL in the *Route Availability Document* [EUR16b]. A comprehensive discussion of the FPP can be found in the survey [Kar+12a]. However, to the best of our knowledge, the algorithmic treatment of flight planning problems on the complete airway network has not yet been considered in the literature. Existing approaches to the FPP, such as [Bon+13b] or [Jon74], consider only small regions of the airway network or artificial networks.

In this paper, we will focus on the *Horizontal Flight Planning Problem* (HFPP), a variant that seeks to minimize total flight time (in this case equivalent to total fuel consumption) while flying at constant altitude. This variant is very important because it is often used in practice as a subroutine in sequential approaches for computing 4-dimensional routes (with speed as the fourth dimension) [Kar+12a]. Furthermore, it can be argued that the cruise phase is more important in terms of potential savings than the climb and descent phases, in particular for long-haul flights. Flight time between any two points is highly dependent on weather conditions, which are given as a function of time. For this reason, we model the HFPP as a Time-Dependent Shortest Path Problem (TDSPP).

The classical Shortest Path Problem (SPP) and the TDSPP have been extensively studied in the literature, with particular emphasis on routing in road networks. The past decades have seen a significant development of preprocessing techniques for both the SPP and the TDSPP, which yield astounding speedups compared to Dijkstra's algorithm, see [Bas+15c], [DW09] for comprehensive surveys. Some of the most prominent state-of-the-art approaches are the following:

- The A* algorithm was first introduced in [HNR68a]. It is based on finding a *potential* function that, for each node, underestimates the length of an optimal path which uses it. The main ingredient for designing a potential function is thus an underestimator of the distance from each node to the target. In road networks, an obvious choice for such an underestimator is the *great circle distance* (GCD) to the target node. However, this method usually provides very loose underestimators (and thus very small speedups), due to the fact that subpaths of the optimal route often deviate substantially from the great circle connecting their endpoints. This can be explained by the grid-like topology of most road networks and the existence of obstacles such as rivers or mountain ranges. Therefore, more sophisticated potential functions have been developed, such as ALT, see the next item.

- *A* with Landmarks and Triangle-inequality* (ALT) [GH04] is a variant of the A* algorithm, which can also be extended to the time-dependent case [Nan+12]. The main idea is to identify a set of "important" nodes, known as *landmarks*, for which a one-to-all (or all-to-one) shortest path tree is computed. The potential of each node is then computed by using these stored distances and the triangle inequality. The main challenge is defining the landmarks, which should ideally lie on a large number of shortest paths, or close to them. The ALT algorithm can be further improved by combining it with other preprocessing techniques, see [Bau+10].

- *Contraction Hierarchies* (CHs) [Gei+12], as well as its time-dependent counterpart, Time-dependent Contraction Hierarchies (TCHs) [Bat+13], is one of the leading techniques in shortest paths computation. Even though TCHs have the disadvantage of large space requirements, they are considered one of the (if not *the*) best algorithms for the TDSPP in road networks [DW09], due to their lower preprocessing times. To the best of

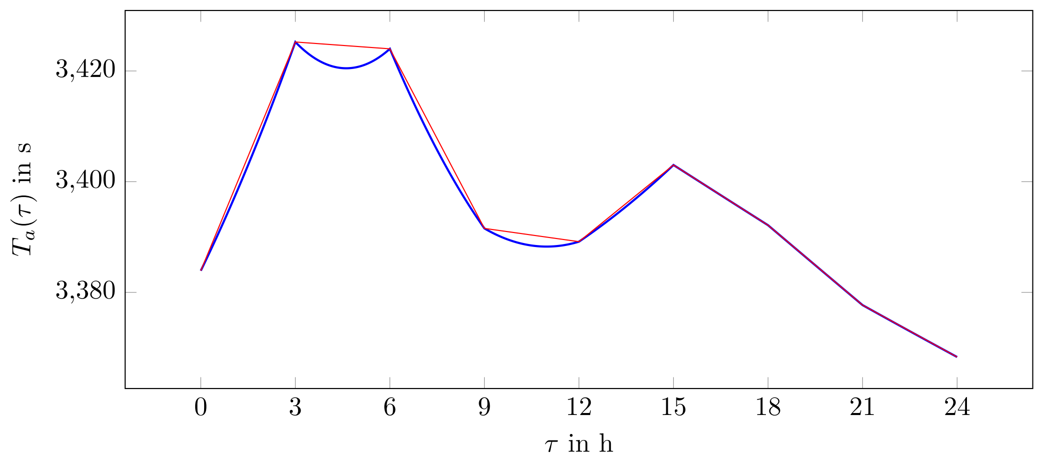


Figure 2.1: The exact travel time function $T(a, \tau)$ in red and the approximated piecewise linear function in blue, for some arc a.

our knowledge, computational results on the performance of CHs and TCHs have been published only for road networks and public transportation networks [Bas+15c].

- Approaches based on *Hierarchical Hub Labeling* [Abr+12] have been shown to be effective not only on road networks but on a large variety of input graphs [Del+14], such as social networks or computer game networks. However, the nature of this approach seems to make it unsuitable for extension to the time-dependent case.

We will consider the real-world airway network. Its characteristics are very different from those of road networks. As of 2016, the complete horizontal network has approximately 53000 nodes and 330000 arcs after some preprocessing (i.e., contracting a large set of nodes with in-degree and out-degree equal to one). The average node degree of over six is higher than in road networks (usually between two and three), but still significantly smaller than in typical social networks (often in the two-digit range, see [Del+14]). An advantage of flight planning over routing in road networks is that the number of possible OD-pairs is small. In fact, only about 1300 airports worldwide are used by commercial airlines[a]. Also, flight paths are typically short, usually involving below one hundred nodes, and do not deviate much from the great circle connection. It turns out that shortest path computation in airway networks is heavily influenced by these characteristics, and that the relative performance of the algorithms is different than in road networks.

In this paper, we investigate three variants of the HFPP.

- The *static* case is a particular shortest path problem, where the nodes belong to a metric space and arc costs are given by the corresponding distance (i.e., the GCD in our case). We will denote this problem as SPP. We present an A* algorithm in which the lower bounds for the potential function are given by the GCD from any node to the target node. This makes it possible to avoid the preprocessing step completely. Our computational results show that the speedup is comparable to that of CHs.

[a]According to data from www.flightradar24.com

- The *exact dynamic* case is a Time Dependent Shortest Path Problem. In contrast to the literature standard, the time-dependent travel time functions (TTFs) on the arcs are not piecewise linear. In fact, in our application, the TTFs depend on wind forecasts and model the exact arc travel time. We refer to this problem as TDSPP-E. We present a *super-optimal wind* algorithm that underestimates the minimal travel time on each arc using the Newton method and establish a strong a priori error bound. The super-optimal wind bounds and the fact that the set of targets is known in advance, allow us to design an A* algorithm that yields a speedup of approximately 20 w.r.t. Dijkstra. Due to the non-linearity of the TTFs, this problem can not be solved by state-of-the-art TDSPP algorithms, in particular TCHs.

- Finally, in the *approximate dynamic* case, we consider a standard Time Dependent Shortest Path Problem. To this purpose, we approximate all TTFs by piecewise linear functions. Figure 2.1 shows an exact TTF and its approximate counterpart. We denote the resulting problem as TDSPP-PWL and present an A* algorithm similar to the one for TDSPP-E. Our computations show that the average speedup is approximately 25 with respect to Dijkstra, and 15 with respect to TCHs.

In Section 2.2, we describe the problems that we will study. In particular, we give a detailed description of the TTFs used in the exact dynamic case, to model the time-dependent influence of the wind on the travel time. Section 3 presents the super-optimal wind algorithm and the corresponding potential functions for the A* algorithm in the exact dynamic case. Finally, Section 4 presents computational results computed on real world data.

2.2 The Horizontal Flight Planning Problem

The HFPP can be modeled in terms of the Time-Dependent Shortest Path Problem, which is defined as follows: Given are a directed graph $D = (V, A)$ (In our application, nodes represent *waypoints* in the airway network and arcs stand for *airway segments*) and, for each $a \in A$, a *travel time function* (TTF) $T(a, \cdot) : [0, \infty) \to [0, \infty)$ that depends on the entering time. The travel time along a path $(v_0, v_1, \ldots, v_k)$ departing at time τ is defined as

$$T((v_0, \ldots, v_k), \tau) = \begin{cases} T((v_0, v_1), \tau) & k = 1 \\ T((v_0, \ldots, v_{k-1}), \tau) + T((v_{k-1}, v_k), T((v_0, \ldots, v_{k-1}), \tau) + \tau) & k > 1. \end{cases}$$

Given a pair of nodes $s, t \in V$ and a departure time $\tau \geq 0$, the objective is to find an s,t-path P in D such that the total travel time $T(P, \tau)$ is minimized.

The literature on the TDSPP usually considers piecewise linear (PWL) TTFs. We will denote this special (*approximate*) case of the dynamic problem as TDSPP-PWL.

The *exact* version of the dynamic problem, which we denote as TDSPP-E, assumes functions $T(a, \cdot)$ as described subsequently in Subsection 2.2.1. Finally, when $T(a, \cdot)$ is constant for every $a \in A$, we obtain the classical shortest path problem, denoted simply as SPP.

A standard assumption on TTFs is that they satisfy the *First-In-First-Out* (FIFO) property, which states that overtaking on arcs is not possible. That is, $T(a, \tau^1) \leq (\tau^2 - \tau^1) + T(a, \tau^2)$ for every $a \in A$, $0 \leq \tau^1 \leq \tau^2$. It is well known that the FIFO property guarantees correctness of the Dijkstra and A* algorithms, while the TDSPP is NP-hard in the general case. In the remainder of this paper, we assume that the FIFO property is always satisfied.

2.2.1 Wind-Dependent Travel Time Functions

In this subsection, we define the travel time functions $T(a, \cdot)$ for the exact dynamic HFPP. We first recall some aeronautics terminology.

Let $a = (u, v) \in A$ and $\tau \geq 0$. For the next definitions, assume that the aircraft is located at u at time τ and then proceeds to traverse a. The *ground distance* $d^G(a)$ is defined as the GCD between u and v, and is thus independent of τ. The *airspeed* v^A is the speed relative to the air mass surrounding the aircraft. In our application, we assume that v^A is constant. Finally, the *ground speed* $v^G(a, \tau)$ is the aircraft's speed relative to the ground at the moment in which the aircraft enters arc a. The ground speed can be described in terms of a wind vector w acting on a at time τ as follows:

$$v^G(a, \tau) = \sqrt{(v^A)^2 - w_C(a, \tau)^2} + w_T(a, \tau).$$

Here, $w_C(a, \tau)$ represents the *crosswind component* and $w_T(a, \tau)$ the *tailwind component* affecting arc a at time τ; these are the components of the wind vector with angles $\frac{\pi}{2}$ and 0 with respect to a's direction, respectively, see Figure 2.2. Since an aircraft's airspeed is always much larger than wind speed, we can assume that v^G is always well-defined and positive.

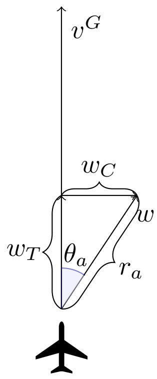


Figure 2.2: Crosswind and tailwind components.

Consider a wind vector $w(a, \tau) = \big(r_a(\tau), \theta_a(\tau)\big)$ acting on a at time τ, where $r_a(\tau)$ is the *wind speed*, i.e., the wind vector's magnitude; and $\theta_a(\tau)$ is the *wind direction*, i.e., the angle with respect to the arc's direction. Then, the crosswind and tailwind components can be computed as follows:

$$w_C(a, \tau) = r_a(\tau)\sin(\theta_a(\tau)) \quad \text{and} \quad w_T(a, \tau) = r_a(\tau)\cos(\theta_a(\tau)).$$

A weather prognosis set provides wind information for a finite number of time points $t_0 < t_1 < \cdots < t_N$. Without loss of generality, we will assume $t_0 = 0$. Furthermore, in practice, prognosis sets are used to plan flights taking off after time t_0 and landing well before time t_N. For that reason, in the rest of the paper, we will assume that we are only interested in evaluating TTFs for $\tau \in [t_0, t_N]$.

If $t_i < \tau < t_{i+1}$ for $i \in \{0, \dots, N-1\}$, the wind vector is interpolated. More precisely, given two wind vectors w_i and w_{i+1} for arc a at times t_i and t_{i+1}, defined by wind speeds r_a^i, r_a^{i+1} and directions $\theta_a^i, \theta_a^{i+1}$, then for $\tau = \lambda t_i + (i - \lambda)t_{i+1}$ with $\lambda \in (0, 1)$, the wind vector at time τ is defined by

$$r_a(\tau) = \lambda r_a^i + (1-\lambda)r_a^{i+1} \quad \text{and} \quad \theta_a(\tau) = \lambda\theta_a^i + (1-\lambda)\theta_a^{i+1}.$$

Therefore the resulting crosswind and tailwind components at time τ are

$$w_C(a, \tau) = \left(\lambda r_a^i + (1-\lambda)r_a^{i+1}\right)\sin\left(\lambda\theta_a^i + (1-\lambda)\theta_a^{i+1}\right)$$
$$w_T(a, \tau) = \left(\lambda r_a^i + (1-\lambda)r_a^{i+1}\right)\cos\left(\lambda\theta_a^i + (1-\lambda)\theta_a^{i+1}\right).$$

In this paper, we assume that $w_C(a, \tau)$ and $w_T(a, \tau)$ remain constant during the traversal of a and define the travel time $T(a, \tau)$ across arc a entering at time τ as

$$T(a, \tau) = \frac{d^G(a)}{v^G(a, \tau)} = \frac{d^G(a)}{\sqrt{(v^A)^2 - w_C(a, \tau)^2} + w_T(a, \tau)}. \tag{2.1}$$

One can argue that the functions $T(a, \cdot)$ in (2.1) satisfy the FIFO property for realistic weather conditions. They represent the industrial state-of-the-art in aeronautical computations.

2.3 A* algorithms for the HFPP

For each of the three problem variants described in Section 2.2, we design an A* algorithm. Such an algorithm is based on a *potential* function $\pi : V \to \mathbb{R}$ that, for every $v \in V$, underestimates the cost of the shortest (v,t)-path. The potential is used to define the *reduced cost* of an arc (u,v) at time τ as follows:

$$T'((u,v),\tau) := T((u,v),\tau) - \pi(u) + \pi(v).$$

If $T'((u,v),\tau) \geq 0$ for every $(u,v) \in A$, $\tau \geq 0$, we say that π is *feasible*. Given this condition, the A* algorithm is equivalent to running Dijkstra on D using the reduced costs T'. In the following, we will introduce potential functions for each of the three problem variants.

2.3.1 Potential in the Static Case

In the static case, i.e., for SPP, a potential function for A* can be computed by simply considering the great-circle-distance between any node and the target node. That is, given $v \in V$ and a target node $t \in V$, we define

$$\pi(v) := d^G(v,t),$$

where $d^G : V \times V \to \mathbb{R}_+$ is the GCD-function. The advantage of this approach is that π can be computed on-the-fly during the query, and so no preprocessing step is necessary.

2.3.2 Potential in the Approximate Dynamic Case

For TDSPP-PWL, we make use of the fact that, in our application, there exists a small number of possible targets (which correspond to airports). Thus, we compute a lower bound on the minimum travel time from each node to each airport. For this, we first seek a value $\underline{T}(a)$ that, for each arc a, lower-bounds all possible travel times on the arc. That is, $\underline{T}(a) \leq T(a,\tau)$ for each $\tau \in [t_0, t_N]$. Since $T(a,\cdot)$ is a piecewise linear function, this bound can be found in linear time. Then, we compute all-to-one shortest path trees towards all airport nodes using $\underline{T}$ as arc costs and set

$$\pi^t(v) = \min \left\{ \sum_{a \in P} \underline{T}(a) | P \text{ is a } (v,t)\text{-path} \right\} \tag{2.2}$$

for every node v and every possible target node t. Given an OD-pair s,t, we choose $\pi^t(\cdot)$ as a potential function. We remark that this is equivalent to choosing all airport nodes as landmarks in the ALT algorithm.

2.3.3 Potential in the Exact Dynamic Case Using Super-Optimal Wind

For TDSPP-E, we also compute lower bounds $\underline{T}$ on the TTFs and then define π according to (2.2). As opposed to the approximate case, finding good lower bounds $\underline{T}(a)$ is not straightforward. This section is dedicated to the solution of this problem.

It is clear from (2.1) that an upper bound on the ground speed directly leads to a lower bound on the travel time. Thus, to find a good lower bound on $T^*(a) := \min_{\tau \in [t_0,t_N]} T(a,\tau)$, we will concentrate on finding a good upper bound on $v^G_*(a) := \max_{\tau \in [t_0,t_N]} v^G(a,\tau)$.

We assume that the length of the weather prognosis intervals is constant, i.e., $t_i - t_{i-1} \equiv L > 0$ for $i = 1, \ldots, N$. Our first step is to discretize the time interval $[t_0, t_N]$ into smaller intervals of length $\Delta > 0$. That is, we define $\tau_0, \ldots, \tau_K$ such that $t_0 = \tau_0 < \tau_1 < \cdots < \tau_K = t_N$,

$\Delta = \tau_k - \tau_{k-1}$ for $k = 1, 2, \ldots, K$; and N divides K. This condition guarantees that every two consecutive time points τ_{k-1} and τ_k belong to an interval $[t_{i-1}, t_i]$ for some index i. Define

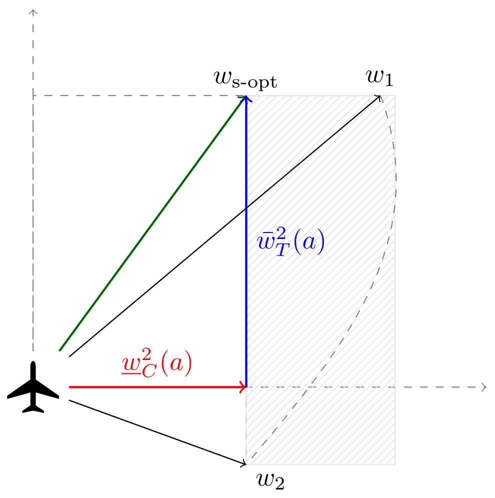
$$\underline{w}_C^k(a) := \min_{\tau \in [\tau_{k-1}, \tau_k]} |w_C(a, \tau)|,$$

$$\bar{w}_T^k(a) := \max_{\tau \in [\tau_{k-1}, \tau_k]} w_T(a, \tau),$$

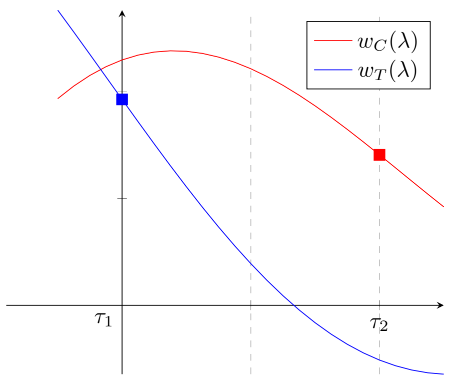
$$\bar{v}_k^G(a) := \sqrt{(v^A)^2 - \underline{w}_C^k(a)^2} + \bar{w}_T^k(a),$$

$$\bar{v}^G(a) := \max_{k \in \{1, \ldots, K\}} \bar{v}_k^G(a),$$

and $\underline{T}(a) := \dfrac{d^G(a)}{\bar{v}^G(a)}$.



(a) The super-optimal wind vector $w_{\text{s-opt}}$ resulting from w_1 and w_2.



(b) Crosswind (red) and tailwind (blue) functions for the wind vectors in the interval $[\tau_1, \tau_2]$

Figure 2.3: Super-optimal wind and component functions for wind vectors w_1 and w_2, corresponding to time points τ_1 and τ_2.

By definition, we know that on any time interval, the ground speed increases as the tailwind increases, and decreases as the crosswind increases. Thus, $(\underline{w}_C^k(a), \bar{w}_T^k(a))$ corresponds to an imaginary *super-optimal wind* vector whose corresponding ground speed $\bar{v}_k^G(a)$ overestimates the ground speed in the time interval $[\tau_{k-1}, \tau_k]$.

For an example, see Figure 2.3. On the right side, we see a typical behavior of the tail- and crosswind functions on an arc in a given time interval $[\tau_1, \tau_2]$, the minimum and maximum of interest are marked. On the left side, we see the super-optimal wind vector that results from the combination of both components. This vector yields a larger ground speed than all wind vectors in the gray rectangle, and thus larger than all wind vectors in the interval $[\tau_1, \tau_2]$, represented by the dashed curve.

From this overestimation property and the definition of $\bar{v}^G(a)$, it follows that, for each arc, the maximum of the ground speed overestimators on all discretization intervals overestimates the ground speed at any time, while the resulting travel time is a global underestimator:

Lemma 2.1. For every $a \in A$ and $\tau \in [t_0, t_N]$, we have

$$\bar{v}^G(a) \geq v_*^G(a) \geq v^G(a,\tau) \quad \text{and} \quad \underline{T}(a) \leq T^*(a) \leq T(a,\tau).$$

Thus, all we need to obtain the bounds $\bar{v}^G(a)$ and $\underline{T}(a)$ is to compute $\underline{w}_C^k(a)$ and $\bar{w}_T^k(a)$ for every $a \in A$, $k = 1, \dots, K$. We will describe that step in Subsection 2.3.4. In the remainder of this subsection, we will prove that the absolute overestimation/underestimation error is linear with respect to the discretization step. Assuming that the aircraft is always at least twice as fast as the wind (which is always the case in practice), we can bound the constant in terms of the airspeed and the length of the weather prognosis intervals.

Theorem 2.1. *For every $a \in A$, assume that $v^A \geq 2r_a^*$. Then, there exists a constant $C^v > 0$ such that the ground speed error is bounded as follows:*

$$0 \leq \bar{v}^G(a) - v_*^G(a) \leq C^v \Delta.$$

Proof. The left inequality follows from Lemma 2.1. For the right one we only have to prove that there exists $C^v > 0$ s.t.

$$\max_{\tau \in [\tau_{k-1}, \tau_k]} \big(\bar{v}_k^G(a) - v^G(a,\tau)\big) \leq C^v \Delta \quad \text{for every } k = 1, 2, \dots, K. \tag{2.3}$$

To bound the ground speed error, we first bound the error on tailwind and crosswind. W.l.o.g assume $k = 1$ and define $I = [\tau_0, \tau_1]$. Let $\rho_1, \rho_2 \in I \subseteq [t_0, t_1]$ and $\lambda_1, \lambda_2 \in [0,1]$ satisfy $\rho_i = \lambda_i t_0 + (1-\lambda_i) t_1$, $i = 1, 2$. We have

$$\begin{aligned} |w_T(a,\rho_1) - w_T(a,\rho_2)| &\leq |\rho_1 - \rho_2| \max_{\rho \in I} |w_T'(a,\rho)| \\ &= |\rho_1 - \rho_2| \max_{\rho \in I} \Big| r_a'(\rho)\cos\big(\theta_a(\rho)\big) - r_a(\rho)\sin\big(\theta_a(\rho)\big)\theta_a'(\rho) \Big| \\ &\leq \Delta\big(\max_{\rho \in I} |r_a'(\rho)| + r_a^* \max_{\rho \in I} |\theta_a'(\rho)|\big), \end{aligned} \tag{2.4}$$

where $r_a^* = \max_{\rho \in [t_0, t_N]} r_a(\rho)$. Since wind speed and direction are interpolated linearly in $[t_0, t_1]$, we have

$$r_a'(\rho) = \frac{r_a^1 - r_a^0}{t_1 - t_0} \quad \text{and} \quad \theta_a'(\rho) = \frac{\theta_a^1 - \theta_a^0}{t_1 - t_0}. \tag{2.5}$$

From (2.4) and (2.5), it follows that

$$|w_T(a,\rho_1) - w_T(a,\rho_2)| \leq \Delta \frac{|r_a^1 - r_a^0| + r_a^* |\theta_a^1 - \theta_a^0|}{t_1 - t_0} \leq \Delta \frac{r_a^*(1 + 2\pi)}{t_1 - t_0}.$$

Similarly, we can prove that

$$|w_C(a,\rho_1) - w_C(a,\rho_2)| \leq \Delta \frac{r_a^*(1 + 2\pi)}{t_1 - t_0}.$$

We are now ready to establish a bound on the ground speed error. Let $\rho^*, \bar{\rho}, \underline{\rho} \in I$ satisfy $\rho^* \in \operatorname{argmax}_{\tau \in I} v^G(a,\tau)$, $w_T(a,\bar{\rho}) = \bar{w}_T^1(a)$, and $w_C(a,\underline{\rho}) = \underline{w}_C^1(a)$. The absolute ground speed error in interval I is thus

$$\begin{aligned}
\bar{v}_1^G(a) - v^G(a,\rho^*) &= \sqrt{(v^A)^2 - w_C(a,\underline{\rho})^2} + w_T(a,\bar{\rho}) - \sqrt{(v^A)^2 - w_C(a,\rho^*)^2} - w_T(a,\rho^*) \\
&= \frac{w_C(a,\rho^*)^2 - w_C(a,\underline{\rho})^2}{\sqrt{(v^A)^2 - w_C(a,\underline{\rho})^2} + \sqrt{(v^A)^2 - w_C(a,\rho^*)^2}} + w_T(a,\bar{\rho}) - w_T(a,\rho^*) \\
&\le \frac{|w_C(a,\rho^*) - w_C(a,\underline{\rho})||w_C(a,\rho^*) + w_C(a,\underline{\rho})|}{\sqrt{(v^A)^2 - r_a(\underline{\rho})^2} + \sqrt{(v^A)^2 - r_a(\rho^*)^2}} + \Delta\frac{r_a^*(1+2\pi)}{t_1 - t_0} \\
&\le \Delta\frac{r_a^*(1+2\pi)}{t_1 - t_0}\left(\frac{r_a(\rho^*) + r_a(\underline{\rho})}{2\sqrt{(v^A)^2 - r_a^{*2}}} + 1\right) \\
&\le \Delta\frac{r_a^*(1+2\pi)}{t_1 - t_0}\left(\frac{r_a^*}{\sqrt{(v^A)^2 - r_a^{*2}}} + 1\right).
\end{aligned}$$

By assumption, the wind speed r_a^* is always smaller than half of the airspeed v^A, so we have

$$r_a^*\left(\frac{r_a^*}{\sqrt{(v^A)^2 - r_a^{*2}}} + 1\right) \le \frac{v^A}{2}(1+1) = v^A.$$

In practice, r_a^* (wind speed) is usually much smaller than $\frac{v^A}{2}$ (flight speed), hence we can choose $C^v := \frac{v^A(1+2\pi)}{L}$. □

Using $\underline{T}(a)\bar{v}^G(a) = d^G(a) = T^*(a)v_*^G(a)$ and Theorem 2.1, the main result of this section follows:

Corollary 2.1. For every $a \in A$, assume that $v^A \ge 2r_a^*$. Then, there exists a constant C^T s.t.

$$0 \le T^*(a) - \underline{T}(a) \le C^T\Delta.$$

That is, assuming reasonable wind conditions, the additive gap between the presented TTF underestimators and the corresponding minima is linearly bounded by the discretization step.

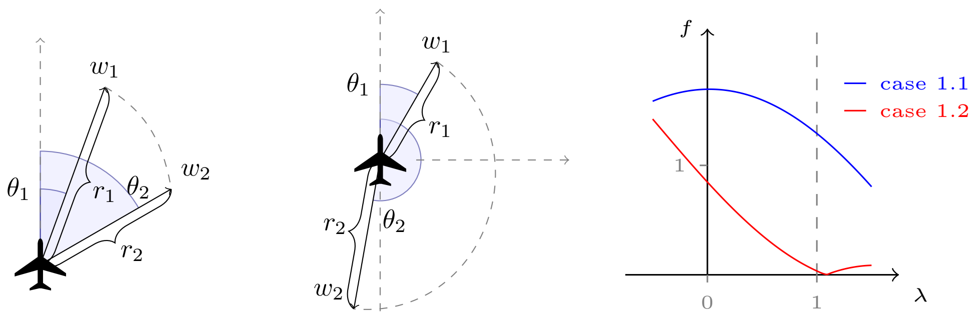
2.3.4 Minimization of Crosswind and Maximization of Tailwind

In the previous subsection, we used the minimum-magnitude crosswind in an interval in order to compute the super-optimal wind vector that is needed to define $\underline{T}(a)$. In this subsection, our objective is to show how this minimization can be done. We recall

$$|w_C(a,\tau)| = |\big(\lambda r_{k-1} + (1-\lambda)r_k\big)\sin\big((\lambda\theta_{k-1} + (1-\lambda)\theta_k\big)|,$$

where $\lambda := \frac{\tau_2 - \tau}{\tau_2 - \tau_1}$ and $\tau \in [\tau_{k-1}, \tau_k]$ for some $k = 1, \dots, K$. W.l.o.g., assume $k = 2$ for ease of notation. It suffices to consider the case $w_C(a,\tau) \ge 0$ for all $\tau \in [\tau_1, \tau_2]$. Indeed, if $w_C(a,\tau)$ takes both positive and negative values in $[\tau_1, \tau_2]$, by continuity the minimum absolute value must be 0, thus making the solution trivial. The case where $w_C(a,\tau) \le 0$ for all $\tau \in [\tau_1, \tau_2]$ is analogous by symmetry. Thus, we can ignore the absolute values.

We can also assume that $\theta_1 < \theta_2$ and $r_1 \ne r_2$, as the other cases are either simple or can be reduced to this case. W.l.o.g. we will further assume that θ_1 and θ_2 belong to the same



(a) θ_1, θ_2 belong to the first quadrant

(b) θ_1, θ_2 belong to different quadrants

(c) Function f in case 1

Figure 2.4: Cases considered for crosswind minimization

quadrant, since otherwise (see e.g. Figure 2.4b) we can compute the minimal value in each quadrant and take the overall minimum. Define

$$w_C(a,\tau) = \big(\lambda r_1 + (1-\lambda) r_2\big) \sin\big((\lambda\theta_1 + (1-\lambda)\theta_2\big) = (a\lambda + b)\sin(\alpha\lambda+\beta) =: f(\lambda)$$

with $a = r_1 - r_2$, $b = r_2 > 0$, $\alpha = \theta_1 - \theta_2 < 0$ and $\beta = \theta_2$. Its derivatives are then

$$\begin{aligned} f'(\lambda) &= a\sin(\alpha\lambda+\beta) + (a\lambda+b)\alpha\cos(\alpha\lambda+\beta),\\ f''(\lambda) &= 2a\alpha\cos(\alpha\lambda+\beta) - \alpha^2(a\lambda+b)\sin(\alpha\lambda+\beta),\\ f'''(\lambda) &= -3a\alpha^2\sin(\alpha\lambda+\beta) - \alpha^3(a\lambda+b)\cos(\alpha\lambda+\beta). \end{aligned}$$

We make the following case distinction:

1. $\theta_1, \theta_2 \in [0, \frac{\pi}{2}]$: We have $\lambda \in [0,1]$, $\sin(\alpha\lambda+\beta), \cos(\alpha\lambda+\beta) \geq 0$. Consider the following two subcases (see Figures 2.4a and 2.4c):

    1.1. $a > 0$, i.e., $r_1 > r_2$: As $(a\lambda + b) > 0$ and since $\alpha < 0$ we have $f''(\lambda) < 0$ for all $\lambda \in [0,1]$. Hence, f is concave and must attain its minimum at either 0 or 1.

    1.2. $a < 0$: Since $f'''(\lambda) > 0$, $f''(\lambda)$ is increasing. Evaluating f'' at $\lambda = 1$ results in two possibilities: If $f''(1) < 0$, we have that f is concave in $[0,1]$, and hence its minimum must be attained at one of the boundary points. If $f''(1) > 0$, we further need to distinguish whether $f''(0) > 0$ (which means f is convex, see below) or $f''(0) < 0$.

    In the latter case, we perform a Newton procedure for finding the inflection point ($f''(\lambda) = 0$), and subdivide $[0,1]$ into its convex and its concave part. Having done so, we know that the minimum in the concave part is attained at one of the end points.

    When f is convex, we apply Newton's method to find a root of $f'(\lambda)$. In case the minimum is found outside of $[0,1]$, we simply take the $\lambda \in \{0,1\}$ closest to it.

    Comparing the values from the concave and convex parts yields the minimum.

2. $\theta_1, \theta_2 \in [\frac{\pi}{2}, \pi]$: We have $\sin(\alpha\lambda+\beta) \geq 0$ and $\cos(\alpha\lambda+\beta) \leq 0$, and again distinguish between two subcases:

    2.1. $a > 0$: Analogous to 1.2.

2.2. $a < 0$: Since $f''(\lambda) < 0$, analogous to 1.1.

3. $\theta_1, \theta_2 \in [\pi, \frac{3\pi}{2}]$: Analogous to 2 by symmetry.

4. $\theta_1, \theta_2 \in [\frac{3\pi}{2}, 2\pi]$: Analogous to 1 by symmetry.

Applying a very similar procedure to the function $g(\lambda)$ defined below yields the maximum of the tailwind component $w_T(a, \tau)$:

$$w_T(a,\tau) = \left(\lambda r_a^1 + (1-\lambda) r_a^2\right) \cos\left(\lambda \theta_a^1 + (1-\lambda)\theta_a^2\right) = (a\lambda + b)\cos(\alpha\lambda + \beta) =: g(\lambda).$$

2.3.5 Validation of Super-Optimal Wind Quality

To assess the quality of the super-optimal wind bounding procedure, we ran it on all arcs in nine instances, corresponding to the three graphs and three weather prognoses described below, in Section 2.4. Our weather prognoses satisfy $L = t_{i+1} - t_i$ equal to three hours for $i = 1, \dots, n$. That, is, precise prognoses are given at three hour intervals and wind is interpolated for times in between. Thus, an obvious candidate for the discretization step Δ is at most three hours. Computational results show that our algorithm with $\Delta = L = 3$h already provides excellent results. To validate our lower bounds, we also used a brute force approach which computes the maximal ground speed on each segment and each time interval through enumeration. The average relative error between our lower bounds and the brute force results is only $0.434 \cdot 10^{-3}$. Also, the average time it takes to process an arc is less than one millisecond; the average run time measured for the complete calculation is 5.61 seconds, running the code on 20 threads on the computer described in Section 2.4. Another interesting fact is that, in almost one third of the cases, the estimated result coincided with the exact result.

2.4 Computational Results

In this section, we present the results of extensive computations measuring the performance of our algorithms on airway networks.

For each of the considered problem variants (SPP, TDSPP-PWL, and TDSPP-E), we implemented a Dijkstra algorithm and an A* algorithm, using the potential functions described in Section 2.3. To test CHs and TCHs on our instances, we used the tools *Contraction Hierarchies* and *KaTCH*, both released by the Karlsruhe Institute of Technology (KIT) [San+16].

Table 2.1: Algorithm nomenclature used in the result tables.

Problem / Algorithm	SPP	TDSPP-PWL	TDSPP-E
Dijkstra	Dijk	Dijk_{PWL}	Dijk_{E}
A*	A*	A^*_{PWL}	A^*_{E}
Contraction Hierarchies	CH	TCH	–

All algorithms (including the Contraction Hierarchies tools) were implemented in C++ and compiled with GCC, and all our computations were performed on computers with 132 GB of RAM and an Intel(R) Xeon(R) CPU E5-2660 v3 processor with 2.60GHz and 25.6 MB cache. All preprocessing steps were carried out in parallel using 20 threads, except for that of static

Contraction Hierarchies, whose code does not offer the option of parallelization. All other computations were carried out in single-thread mode.

We use the notation introduced in Table 2.1 to refer to the different algorithms. In all subsequent tables, we use the abbreviations "prep" for preprocessing time, "query" for query time (given an OD pair) and "speedup" for the ratio between the given algorithm's query times and Dijkstra's query times.

2.4.1 Instances

All instances used in our computations correspond to real-world data, provided to us either by Lufthansa Systems GmbH & Co. KG (graphs and weather prognoses) or obtained from the flight tracking portal www.flightradar24.com (list of OD pairs).

We consider three directed graphs, corresponding to horizontal layers of the airway network at altitudes 29000 feet, 34000 feet, and 39000 feet, respectively. We chose these particular three because they are all common cruise altitudes distant enough from each other that the weather conditions are substantially different. While the graphs are topologically very similar, there exist several arcs which may be used only at certain altitudes. The characteristics of the three graphs and the notation we will use to refer to them are presented in Table 2.2.

Table 2.2: Graph instances corresponding to three common flight altitudes.

Instance	Nodes	Arcs	Avg. degree	Flight altitude
I-29	52719	329442	6.249	29000ft
I-34	52691	329736	6.258	34000ft
I-39	52662	329580	6.258	39000ft

Furthermore, we consider three different sets of weather prognoses. Each contains weather information for a period ranging from 30 to 45 hours, with prognoses available at intervals of 3 hours. We identify them by the names *Dec*, *Feb* and *Mar*, based on the dates in which the prognoses were made.

Table 2.3: Comparison of CH and A* in the static case.

	Dijk	CH			A*		
Instance	query (ms)	prep (s)	query (ms)	speedup ×	prep (s)	query (ms)	speedup ×
I-29	2.01	1260	0.37	**5.45**	0	0.34	5.86
I-34	2.00	1233	0.38	5.24	0	0.33	**6.12**
I-39	1.94	1309	0.39	5.01	0	0.32	6.00

To construct instances for TDSPP-PWL, we approximated the TTFs with piecewise linear functions by discretizing the time horizon into time intervals of length one/three hours. Three hours is an obvious choice, since each prognosis set makes predictions for time points at three hours intervals. All TTFs thus obtained satisfy the FIFO property.

For all algorithms, we ran queries on a fixed set of 18644 OD pairs, corresponding to all flights recorded by www.flightradar24.com in June, 2015.

We use the following notation to identify our instances. For SPP, instances are given by the altitude (e.g. I-29). For TDSPP-E, instances are defined by the altitude and the weather prognosis set (e.g. I-29-Feb). Finally, for TDSPP-PWL, we identify instances by the altitude, prognosis, and discretization size (e.g. I-29-Feb-3).

Table 2.4: Comparison of TCHs and A^*_{PWL} for TDSPP-PWL

Instance	DIJK_{PWL} query (ms)	TCH prep (min)	TCH query (ms)	TCH speedup ×	A^*_{PWL} prep (s)	A^*_{PWL} query (ms)	A^*_{PWL} speedup ×
I-29-Dec-1	4.91	380.48	4.08	1.20	1.82	0.22	21.51
I-34-Dec-1	4.91	451.82	4.27	1.15	1.83	0.24	20.24
I-39-Dec-1	4.93	195.75	3.23	1.53	1.81	0.16	30.15
I-29-Feb-1	4.90	414.78	3.94	1.25	1.87	0.21	22.96
I-34-Feb-1	4.86	466.95	3.96	1.23	1.72	0.21	22.23
I-39-Feb-1	4.92	184.20	3.01	1.63	1.72	0.15	31.50
I-29-Mar-1	4.55	216.57	2.82	1.61	1.50	0.16	27.27
I-34-Mar-1	4.55	189.18	2.92	1.55	1.56	0.18	24.38
I-39-Mar-1	4.58	127.38	2.52	1.81	1.54	0.15	29.45
I-29-Dec-3	4.36	312.40	2.67	1.63	1.54	0.19	22.03
I-34-Dec-3	4.38	351.70	2.80	1.56	1.54	0.21	20.85
I-39-Dec-3	4.38	160.20	2.30	1.90	1.54	0.14	30.87
I-29-Feb-3	4.31	328.47	2.66	1.62	1.51	0.18	23.09
I-34-Feb-3	4.28	372.15	2.92	1.47	1.60	0.19	21.68
I-39-Feb-3	4.33	155.07	2.20	1.97	1.52	0.13	**31.94**
I-29-Mar-3	4.22	179.45	2.31	1.82	1.34	0.14	28.39
I-34-Mar-3	4.26	146.52	2.33	1.83	1.37	0.16	26.68
I-39-Mar-3	4.26	96.80	2.03	**2.10**	1.35	0.13	31.02
Summary							
Average	4.55	262.77	2.94	1.60	1.59	0.18	25.90
Minimum	4.22	96.8	2.03	1.15	1.34	0.13	20.24
Maximum	4.93	466.95	4.27	2.10	1.87	0.24	31.94

2.4.2 Static Case

The results for SPP can be found in Table 2.3. The speedup obtained by A* is slightly better than that of CHs, but not significantly. What is remarkable is that, since the potential for A* is computed on-the-fly during query time, no preprocessing is necessary. This results in a distinct advantage over CHs, which require over 20 min. preprocessing time. However, this is also the reason why the query times of A* are longer than in the time-dependent version (see Table 2.4). In fact, the computation of the potential functions accounts for over half the CPU time needed for the queries. The good performance of A* in this case is likely due to the fact that airway networks allow for minimum-distance paths to lie close to the great circle, as opposed to road networks.

2.4.3 Approximate Dynamic Case

In Table 2.4, we compare the results for the solution of TDSPP-PWL: A^*_{PWL} is the clear winner. We can see that the preprocessing time of TCHs is much longer than that of A*, and is in fact too long to be of use in practical applications. Furthermore, the query times of A^*_{PWL} yield an approximate speedup of 25 w.r.t. Dijkstra and 15 w.r.t. TCHs. Recall that A^*_{PWL} can exploit the fact that the set of possible target nodes is small and known in advance, while TCHs have no such advantage. This partially explains the former algorithm's superiority.

2.4.4 Exact Dynamic Case

Finally, in Table 2.5, we compare our versions of A* implemented for TDSPP-E and TDSPP-PWL. While $A^*_{PWL_1}$ and $A^*_{PWL_3}$ refer to the same algorithm, we use the indices 1 and 3 to distinguish between the instances with corresponding discretization steps. The preprocessing times measured for $A^*_{PWL_1}$ and $A^*_{PWL_3}$ include both the construction of the piecewise linear

Table 2.5: Comparison of A^*_E and A^*_{PWL} in the time-dependent case.

Instance	DIJK$_E$	A^*_E			$A^*_{PWL_1}$				$A^*_{PWL_3}$			
	query (ms)	prep (s)	query (ms)	speedup ×	prep (s)	av err (%)	max err (%)	bad paths (#)	prep (s)	av err (%)	max err (%)	bad paths (#(%))
I-29-Dec	100.89	7.51	5.80	17.38	139.41	0.059	**8.76**	506 (2.71%)	46.69	0.078	8.76	740 (3.97%)
I-34-Dec	102.12	7.38	6.13	16.64	140.81	0.072	5.30	701 (3.76%)	47.88	0.093	**10.92**	940 (5.04%)
I-39-Dec	104.33	7.56	4.47	23.34	140.98	0.018	2.65	79 (0.42%)	47.93	0.021	2.65	94 (0.50%)
I-29-Feb	100.88	7.66	5.49	18.37	139.74	0.028	5.38	195 (1.05%)	47.03	0.035	5.38	269 (1.44%)
I-34-Feb	101.37	7.35	5.68	17.85	141.32	0.038	4.64	317 (1.70%)	48.43	0.049	4.63	431 (2.31%)
I-39-Feb	104.44	7.45	4.16	**25.09**	140.72	0.015	3.60	51 (0.27%)	48.45	0.019	3.60	75 (0.40%)
I-29-Mar	100.07	7.14	4.85	20.60	91.38	0.022	5.37	96 (0.51%)	31.26	0.030	5.41	183 (0.98%)
I-34-Mar	35.72	5.77	1.85	19.25	92.78	0.019	4.60	87 (0.47%)	32.34	0.022	4.60	111 (0.60%)
I-39-Mar	36.18	5.68	1.59	22.66	95.21	0.016	4.74	89 (0.48%)	33.01	0.017	4.74	93 (0.50%)
Summary												
Average	87.33	7.06	4,45	20.13	124.71	0.032	5.00	235.67 (1.26%)	42.56	0.040	5.63	326.22 (1.75%)
Minimum	35.72	5.68	1,59	16.64	91.38	0.015	2.65	51.00 (0.27%)	31.26	0.017	2.65	75.00 (0.40%)
Maximum	104.44	7.66	6,13	25.09	141.32	0.072	8.76	701.00 (3.76%)	48.45	0.093	10.92	940.00 (5.04%)

functions (not considered in Table 2.4, since in that case the procedure is needed by all algorithms) and of the potential functions. Comparing the query times with those of $A^*_{PWL_1}$ and $A^*_{PWL_3}$ (Table 2.4) shows that the running time increases by a factor of over 20. This is a disadvantage of the exact method, even though it is still very fast.

On the other hand, measuring the impact of the approximations on the final solution reveals some outliers. To this purpose, we compare the optimal solution returned by Dijkstra with that returned by the A^*_{PWL} algorithms. For both solutions we compute the exact minimal travel time and the PWL objective value. Table 2.5 displays the average relative error, the maximum relative error, and the number of "bad paths", which are defined as OD pairs for which the relative error is larger than 0.5%. This value is interesting since, in the flight planning industry, savings of 0.5% can justify longer running times. The number of bad paths is not insignificant, and justifies the consideration of the exact method.

2.5 Conclusion

This paper shows that airway networks allow significant speedups in shortest path computations over Dijkstra's algorithm, but with different methods than those used for road networks. In particular, it turns out that the A^* algorithm with problem-specific potentials performs better than Contraction Hierarchies. We discuss three different versions of the Horizontal Flight Planning Problem: The shortest path problem with static costs, the time-dependent shortest path problem with piecewise-linear TTFs, and a special case of the time-dependent shortest path problem with non-piecewise-linear (weather-dependent) TTFs.

For the first two variants, A^* potentials based on GCDs and PWL approximations, respectively, are faster than CHs and TCHs. In both cases, the preprocessing time needed by A^* is shorter than that of CHs by several orders of magnitude. In the static case, the query times of both algorithms are comparable, while in the case of piecewise linear TTFs, A^* outperforms TCHs by a factor of 15. It remains an open question whether Contraction Hierarchies can be adapted to attain a better performance on airway networks.

For the variant of non-piecewise-linear TTFs, we propose a *super-optimal wind* procedure for underestimating TTFs. We present tight theoretical and empirical bounds on its approximation error. The A^* algorithm resulting from these bounds yields a speedup factor of 20 with respect to Dijkstra and very short preprocessing times. We also analyze the effect of approximating TTFs with piecewise linear functions. This approximation approach leads to extremely fast query times and a very small average error, but produces a few outliers. An interesting research direction is to combine the advantages of these methods.

Future research also includes adapting these techniques for the three-dimensional flight planning problem. This is not straightforward since the TTFs corresponding to climb and descent phases depend not only on the wind, but also on the current aircraft's weight and technical specifications.

References

[Abr+12] Ittai Abraham, Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck. "Algorithms – ESA 2012: 20th Annual European Symposium, Ljubljana, Slovenia, September 10-12, 2012. Proceedings". In: ed. by Leah Epstein and Paolo Ferragina. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. Chap. Hierarchical Hub Labelings for Shortest Paths, pp. 24–35.

[Bas+15c] Hannah Bast, Daniel Delling, Andrew V. Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. "Route Planning in Transportation Networks". In: *CoRR* abs/1504.05140 (2015).

[Bat+13] G. Veit Batz, Robert Geisberger, Peter Sanders, and Christian Vetter. "Minimum Time-dependent Travel Times with Contraction Hierarchies". In: *Journal of Experimental Algorithmics* 18 (Apr. 2013), 14:11–14:143. ISSN: 1084-6654.

[Bau+10] Reinhard Bauer, Daniel Delling, Peter Sanders, Dennis Schieferdecker, Dominik Schultes, and Dorothea Wagner. "Combining Hierarchical and Goal-directed Speed-up Techniques for Dijkstra's Algorithm". In: *Journal of Experimental Algorithmics* 15 (Mar. 2010), 23:21–23:231. ISSN: 1084-6654.

[Bon+13b] Pierre Bonami, Alberto Olivares, Manuel Soler, and Ernesto Staffetti. "Multiphase Mixed-Integer Optimal Control Approach to Aircraft Trajectory Optimization". In: *Journal of Guidance, Control, and Dynamics* 36.5 (July 2013), pp. 1267–1277. ISSN: 0731-5090. DOI: 10.2514/1.60492.

[Del+14] Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck. *Robust Exact Distance Queries on Massive Networks.* Tech. rep. July 2014.

[DW09] Daniel Delling and Dorothea Wagner. "Robust and Online Large-Scale Optimization: Models and Techniques for Transportation Systems". In: ed. by Ravindra K. Ahuja, Rolf H. Möhring, and Christos D. Zaroliagis. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. Chap. Time-Dependent Route Planning, pp. 207–230. ISBN: 978-3-642-05465-5.

[EUR16b] EUROCONTROL. *Route Availability Document.* [Online; accessed 2-June-2016]. 2016. URL: `https://www.nm.eurocontrol.int/RAD/`.

[Gei+12] Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. "Exact Routing in Large Road Networks Using Contraction Hierarchies". In: *Transportation Science* 46.3 (Aug. 2012), pp. 388–404. ISSN: 1526-5447.

[GH04] A. V. Goldberg and C. Harrelson. *Computing the Shortest Path: A* Search Meets Graph Theory.* Tech. rep. Vancouver, Canada, July 2004.

[HNR68a] P. E. Hart, N. J. Nilsson, and B. Raphael. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (July 1968), pp. 100–107. ISSN: 0536-1567.

[Jon74] H.M. de Jong. *Optimal Track Selection and 3-dimensional flight planning.* Tech. rep. 1974.

[Kar+12a] Stefan E. Karisch, Stephen S. Altus, Goran Stojković, and Mirela Stojković. “Operations”. English. In: *Quantitative Problem Solving Methods in the Airline Industry.* Ed. by Cynthia Barnhart and Barry Smith. Vol. 169. International Series in Operations Research & Management Science. Springer US, 2012, pp. 283–383. ISBN: 978-1-4614-1607-4.

[Nan+12] Giacomo Nannicini, Daniel Delling, Dominik Schultes, and Leo Liberti. “Bidirectional A* Search on Time-dependent Road Networks”. In: *Networks* 59.2 (Mar. 2012), pp. 240–251. ISSN: 0028-3045.

[San+16] Peter Sanders, Moritz Kobitzsch, Veit Batz, Robert Geisberger, Dennis Luxen, Dennis Schieferdecker, Dominik Schultes, and Christian Vetter. *Fast and Exact Route Planning.* [Online; accessed 2-June-2016]. 2016. URL: http://algo2.iti.kit.edu/routeplanning.php.

Introduction to Chapter 3

This chapter was published in [SMB19].

It relies on the Super-Optimal-Wind underestimators presented in Chapter 2 and on the Two-Layer-Dijkstra algorithm presented in Chapter 5 in order to achieve significant search space reductions via *pruning*. As opposed to the A* algorithms presented in Chapters 2 and 4, this reduction is done prior to the search rather than during it.

One reason for doing it this way is that, at the time of doing this work, we did not know how to efficiently calculate good fuel underestimators starting from arbitrary altitudes (which is necessary for an A* algorithm) and had to rely on doing so for complete routes (for which we can assume a fixed starting altitude) on fixed distances. This aspect was improved with the results presented in Chapter 4.

But the main advantage of this approach is that it allows the use of search algorithms different than Dijkstra and A*. In particular, if the underlying directed graph is acyclic, one can use a topological sorting based algorithm. Not only does this eliminate the need for a priority queue, but such an algorithm is very suitable for parallelization. In the context of flight planning, it is a reasonable assumption that the underlying graph will be acyclic, as commercial flights typically do not fly "backwards" towards the departure airport or away from the destination airport.[b] This makes this approach particularly well-suited for solving the shortest path problem in this domain, potentially resulting in an algorithm faster than one based on a priority queue.

One of the key contributions of this paper is the introduction of an efficient search algorithm for flight planning in the three-dimensional space, taking into account weather, fuel consumption, and overflight charges. To the best of our knowledge, this is the first algorithm that considers all these aspects and is adequate for practical use. [Bon+13b] includes these three factors in its model, but assumes constant altitude and does not provide the level of performance and practicality needed for real-world applications. Similarly, [KCL18] addressed most of these issues and is very efficient, but did not consider overflight charges. In the interest of fairness, we should note that it does consider traffic restrictions, which are absent from this chapter.

References

[Bon+13b] Pierre Bonami, Alberto Olivares, Manuel Soler, and Ernesto Staffetti. "Multiphase Mixed-Integer Optimal Control Approach to Aircraft Trajectory Optimization".

[b]While departing from an airport or approaching one, it is indeed common for planes to take detours. But this is because they follow pre-defined arc chains called terminal procedures, which connect the airports to the airway network. These are, by design, never part of cycles.

In: *Journal of Guidance, Control, and Dynamics* 36.5 (July 2013), pp. 1267–1277. ISSN: 0731-5090. DOI: `10.2514/1.60492`.

[KCL18] Anders Nicolai Knudsen, Marco Chiarandini, and Kim S. Larsen. “Heuristic Variants of A* Search for 3D Flight Planning”. In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 15th International Conference, CPAIOR 2018, Delft, The Netherlands, June 26-29, 2018, Proceedings.* 2018, pp. 361–376. DOI: `10.1007/978-3-319-93031-2_26`. URL: `https://doi.org/10.1007/978-3-319-93031-2_26`.

[SMB19] Adam Schienle, Pedro Maristany, and Marco Blanco. “A Priori Search Space Pruning in the Flight Planning Problem”. In: *19th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2019).* Ed. by Valentina Cacchiani and Alberto Marchetti-Spaccamela. Vol. 75. OpenAccess Series in Informatics (OASIcs). This work is licensed under the Creative Commons Attribution 3.0 Unported License. To view a copy of this license, visit `http://creativecommons.org/licenses/by/3.0/`. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, 8:1–8:14. ISBN: 978-3-95977-128-3. DOI: `10.4230/OASIcs.ATMOS.2019.8`.

Chapter 3

A Priori Search Space Pruning in the Flight Planning Problem

Adam Schienle, Pedro Maristany, Marco Blanco

Abstract

We study the Flight Planning Problem for a single aircraft, where we look for a minimum cost path in the airway network, a directed graph. Arc evaluation, such as weather computation, is computationally expensive due to non-linear functions, but required for exactness. We propose several pruning methods to thin out the search space for Dijkstra's algorithm before the query commences. We do so by using innate problem characteristics such as an aircraft's tank capacity, lower and upper bounds on the total costs, and in particular, we present a method to reduce the search space even in the presence of regional crossing costs.

We test all pruning methods on real-world instances, and show that incorporating crossing costs into the pruning process can reduce the number of nodes by 90% in our setting.

3.1 Introduction

With a looming climate change and an ever more connected world, it is imperative that aircraft routes are planned as efficient and efficiently as possible. Contrary to popular belief, aircraft cannot fly directly between their origin and destination airports, but have to adhere to the *Airway Network*, a directed graph. The nodes of the graph are called *waypoints*, whereas the arcs are called *(airway) segments*. For vertical separation, aircraft are stacked on 43 *flight levels*, which are mostly spaced 1 000 ft apart.

Airlines plan a flight by computing an optimal route according to their preferences, which may include minimum fuel, minimum time, minimum cost (e.g., in USD), or a combination thereof. Furthermore, one has to consider various overflight charges for countries, weight dependent fuel consumption functions and weather-dependent arc lengths. Due to the weather being time-dependent, this introduces an implicit time-dependency into the problem. On top, some air navigation service providers publish restrictions to prevent congestion, such as EUROCONTROL's *Route Availability Document*[Eur19b].

The **Flight Planning Problem** as we will discuss it is the problem of finding a minimum cost (in USD) trajectory given an origin and a destination airport, a departure time and a weather forecast, an aircraft and its consumption functions, and overflight charges for each country. Note that we take the airline's point of view, and only consider one aircraft at a time. A general introduction to this problem can for instance be found in [Kar+12b]. We will in the course of this paper disregard restrictions imposed by air navigation service providers (ANSPs), as they can usually only be enforced on a given route. In particular, the Flight Planning Problem as we see it is a base problem which may have to be solved a number of times in order to obtain a valid flight plan.

In practice, a flight plan is compiled a few hours before the aircraft takes off. It is then filed with ANSPs such as EUROCONTROL, who either accept or reject it. If rejected, it must be recomputed to comply with additional restrictions; if accepted, however, it must be flown as is, which requires the flight plan to be optimal – and discourages approximative algorithms. Since flight plans may have to be recomputed following a rejection by ANSPs, and since the base problem may have to be re-solved in order to obtain a solution which satisfies restrictions, it is necessary that the base problem can be solved fast.

In this paper, we aim to reduce the a priori search space before the query commences. To do so, we use a combination of upper and lower bounds in order to remove nodes from the graph which provably cannot lie on an optimal path. Since many countries (especially in Europe) allow more and more direct connections (so-called Free Route Areas) between any two nodes within their boundaries, the underlying graph tends to become denser in the future, which negatively affects runtimes.

3.1.1 Literature and Related Work

Reliant on a directed graph, the Flight Planning Problem shares similarities with the (Time-Dependent) Shortest Path Problem on road networks. The best-known algorithm to solve the non time-dependent version is Dijkstra's Algorithm[Dij59], which can be extended for the time-dependent setting, see [Dre69]. However, the time-dependent version of Dijkstra's Algorithm is only guaranteed to find an optimal solution if the FIFO property is satisfied; we say that a function $f\colon A \times \mathbb{R}_0^+$ satisfies the FIFO property, if

$$\tau < \tau' \Rightarrow f(a, \tau) \leq f(a, \tau') + (\tau' - \tau).$$

This property basically states that overtaking is not possible by waiting at nodes. Many speedup techniques for Dijkstra's Algorithm exist for both the static and the time-dependent version; [Bas+15a] provides a good overview. While some such as A*[HNR68b] use potential functions to guide the query at runtime, the methods which are most effective on road networks require one or more preprocessing step. Contraction Hierarchies[Gei+12] (CHs) and its time-dependent sibling Time-Dependent Contraction Hierarchies[Bat+13] (TCHs) assign ranks to the nodes, and look for a shortest path through an in- and then decreasing sequence of nodes.

In [Bla+16a], the authors show that TCHs do not perform as effectively on the airway network graph as on road networks, being dominated both in preprocessing and in query times by A*. The authors also introduce a novel technique for underestimating traversal times in the flight planning problem. However, they do not consider overflight charges, which are a central component of this paper.

Jensen et al. introduce a geometric algorithm to solve the Free Flight Problem[JCL17a]. They partition any free flight airspace into rectangles of equal, constant wind. While similar to Dijkstra's algorithm, their method sorts nodes based not on their costs, but on the costs of their cheapest successor, and lets nodes compete among each other for their successors, thus achieving a speedup over Dijkstra's Algorithm or A*. One has to note, however, that their approach assumes airspace users can choose their waypoints freely. On the other hand, we will assume that all waypoints and segments are defined, and that one is not allowed to fly via self-defined waypoints. This represents the current point of view as expressed by air navigation service providers such as EUROCONTROL[Eur19a].

In terms of overflight charges or crossing costs, relatively little research has been conducted. Most notably, Blanco et al. define and analyse the Shortest Path Problem with Crossing Costs in [Bla+16b], and solve a special case to optimality. In [Bla+17a], the authors furthermore propose a cost projection onto the arcs. Dreves and Gerdts [DG17] give an example on how to solve the problem using optimal control, albeit in a bounded region in Europe. None of the works cited deal with overflight charges considers the influence of weather. Both [Bla+16a] and [Bla+17a] only run queries on one layer of the airway network, whereas we use the full (3D) graph for our computations.

We seek to cut nodes with several different methods from the a-priori search space, pruning all those nodes which provably cannot lie on an optimal path. This technique is also used in other applications of shortest path problems, e.g. in Electric Vehicle Routing[Bau+14].

Crossing costs pose a particular problem, in that they do not always correlate to the arcs or nodes on the path, but rather depend on geometric information given by entry and exit points for the given regions. While all of the presented pruning methods bear similarities to A*, to the best of our knowledge, there is no known underestimator for regional crossing costs of this particular type.

In section 3.2, we develop the theory on how to prune nodes prior to the query. Section 3.3 shows how to model the Free Flight Problem as a Time-Dependent Shortest Path Problem with Crossing Costs. We develop several different pruning methods for the Free Flight Problem in section 3.4, and show the results of real-world test cases in section 3.5.

3.2 Pruning Search Spaces in the Time-Dependent Shortest Path Problem

In this section, we consider the Time-Dependent Shortest Path Problem (TDSPP) defined as follows: We are given a graph $G = (V, A)$ together with a travel time function $T\colon A \times \mathbb{R}_0^+ \to \mathbb{R}_0^+$, which maps an arc a and the time τ at which it is entered to the **travel time** $T(a, \tau)$ needed to traverse a. This function T will be one constituent of the total cost. For a path $P = (v_0, \dots, v_n)$ between two nodes v_0, v_n and a departure time τ_0, we define the travel time for P as

$$T(P, \tau_0) := \sum_{i=0}^{n-1} T\big((v_i, v_{i+1}), \tau_i\big),$$

where $\tau_{i+1} = \tau_i + T\big((v_i, v_{i+1}), \tau_i\big)$ for every $i \in \{0, \dots, n-1\}$. We further impose that two nodes $s, t \in V$ exist, and seek to find the shortest path from s to t with respect to T, starting at time τ_0.

In [Bla+16a], the authors show that in flight planning, A* outperforms even the most promising speedup technique to Dijkstra's Algorithm, Time-Dependent Contraction Hierarchies[Bat+13]. Since our pruning techniques in section 3.4 are similar to A*, we limit ourselves to the discussion of pruning the search space for Dijkstra's algorithm.

In this section, we concentrate on how to prune the search space for Dijkstra's algorithm before the query begins. To this end, we use both lower bounds on the arc costs as well as an upper bound on the route costs. Usually, computing an upper bound on the route costs is easy by just computing any feasible solution.

Let P^* be a minimum cost path s-t path starting at τ_0, and $c^* := c(P^*)$ its cost. We assume that we are given an upper bound $\bar{c}$ on c^*, e.g., through a previously computed feasible solution.

Theorem 3.1. *Assume we are given a* TDSPP *as defined above, and an upper bound $\overline{c}$ on the costs for a shortest path between s and t. Let $\underline{c}_s^t \colon V \to \mathbb{R}_0^+$ be a function which underestimates the costs for a shortest s-t-path via v. Any node $v \in V$ which violates the pruning inequality*

$$\underline{c}_s^t(v) \leq \overline{c}$$

cannot lie on an optimal path.

Proof. Let P^* be an optimal path, and assume that $v \in P^*$ violates the pruning inequality, so there exists a shortest s-t-path P_v via v such that $\underline{c}(P_v) > \overline{c}$. Then we find

$$c^* = c(P^*) \geq \underline{c}(P_v) > \overline{c},$$

contradicting the assumption that P^* was optimal to begin with. □

Remark 3.1. In practice, we obtain a function $\underline{c}_s^t(v)$ as required in the above theorem by computing a lower bound function $\underline{c} \colon A \to \mathbb{R}_0^+$ such that

$$\underline{c}(a) \leq c(a, \tau) \quad \forall \tau \in \mathbb{R}_0^+.$$

This approach is common for shortest path problems (e.g. for A* or ALT, see [Bas+15a]), and is also used in [Bla+16a]. We write $\underline{G} := (G, \underline{c})$, and run a one-to-all Dijkstra from s and an all-to-one Dijkstra to t on $\underline{G}$. Note that $\underline{G}$ carries static arc costs, hence we can grow the Dijkstra trees in $\mathcal{O}(|A| + |V| \log |V|)$.

If any node $v \in V$ is not contained in either of the two trees, it cannot be reached from s, or cannot reach t. In both cases, it can safely be eliminated, as it cannot lie on any optimal path. If a node $v \in V$ is not contained in both the forward and the backward tree, we will assume its respective costs to be ∞.

Theorem 3.1 states that any path whose lower-bound costs are already higher than a pre-computed upper bound cannot be optimal. Note that this mimics the A* algorithm[HNR68b]. The key difference is that by using Theorem 3.1, nodes which cannot lie on an optimal path are eliminated a priori, rather than being discarded during the search. This means that it is a little weaker than A*: for any node $v \in V$, the latter adds an estimate of the remaining costs from v to t to the actual costs $c(P_s^v)$ from s to v, and sorts the node in the heap accordingly. Contrastingly, when applying Theorem 3.1, one adds two cost estimates and compares them to an upper bound. Unless the estimate is perfect, this leads to a gap between $\underline{c}(P_s^v)$ and $c(P_s^v)$. However, in practice it might be easier to compute an underestimation for the costs of a complete path, rather than just parts of it. Also, a pre-pruned search space can be advantageous for search algorithms which do not maintain a heap structure; e.g., by sorting the graph's nodes in topological order and run a search algorithm exploring the resulting node groups in their respective order.

Remark 3.2. The applicability of Theorem 3.1 is dependent on both the quality of the lower bound as well as the quality of the upper bound solution. Clearly, lowering the upper bound will result in more nodes being eliminated, as will raising the lower bound.

As shown by Fredman et al.[FT87], the runtime for the static Dijkstra's Algorithm is $\mathcal{O}(|A| + |V| \log |V|)$ when a Fibonacci heap is used to store the unprocessed labels. The dynamic, i.e., time-dependent case with FIFO travel time functions can be solved using almost the same version of the algorithm [Dre69]. The only difference is the evaluation of the arcs' cost: in the static case it can be in constant time but in the dynamic case the complexity of the evaluation depends on the shape of the functions [FHS14]. This is why, even in the FIFO case, it is not guaranteed that the TDSPP is polynomially solvable and this is also the motivation for a restriction of the search space before the query commences.

3.3 Modelling the Free Flight Problem

We model the Free Flight Problem as a **Time-Dependent Shortest Path Problem with Crossing Costs**, or T-SPROC for short. As the name suggests, we introduce crossings costs to the Time-Dependent Shortest Path Problem to account for overflight charges. We proceed as follows:

We consider the airway network as directed graph $G = (V, A)$, and origin and destination airport as distinct nodes s and t in V. Each flight starts at a departure time $\tau_0 \in \mathbb{R}_0^+$. In our application, the cost functions comprises three components, to wit, the *fuel costs*, the *time costs*, and *overflight costs*. Fuel costs are defined by what an aircraft burns en route, while overflight costs are charges raised by countries' air navigation service providers. Time costs, on the other hand, comprise leasing costs, crew costs, and maintenance costs, and can in our case be considered a linear function of the time en route. Hence, we can think of time costs as being charged per arc, and introduce the *time cost function*

$$\begin{aligned} c_t \colon A \times \mathbb{R}_0^+ &\to \mathbb{R}_0^+ \\ (a, \tau) &\mapsto \alpha \cdot T(a, \tau). \end{aligned}$$

Fuel costs depend linearly on how much fuel an aircraft burns en route. The fuel burn is directly proportional to the distance relative to the surrounding air mass, or *air distance*. As in [JCL17a; Bla+16a], we assume that an aircraft flies with constant speed, which in turn renders the air distance proportional to the time en route. Therefore, we can again think of fuel costs as being charged per arc, and write

$$\begin{aligned} c_f \colon A \times \mathbb{R}_0^+ &\to \mathbb{R}_0^+ \\ (a, \tau) &\mapsto \beta \cdot T(a, \tau). \end{aligned}$$

Both fuel and time costs naturally depend on the travel time, which in turn is weather-dependent. We use the same travel time functions as given [Bla+16a], assuming that weather is given for a discrete point set $\Delta \subset \mathbb{R}_0^+$ in time over a long enough interval to cover all flight durations. Usually, the break points are three hours apart; for any $\tau \notin \Delta$, we interpolate the closest two weather data objects to obtain a wind vector $\mathbf{w}(a, \tau)$ for an arc $a \in A$. It can be decomposed into its cross wind component $w_c(a, \tau)$ perpendicular to the direction of flight, and a track wind component $w_t(a, \tau)$ parallel to it. Together with the constant air speed, this leads to the travel time (c.f. [Bla+16a; JCL17a])

$$T(a, \tau) = \frac{\ell(a)}{\sqrt{v^2 - w_c^2(a, \tau)} + w_t(a, \tau)},$$

with $\ell(a)$ denoting the length over ground of the arc $a \in A$. In particular, our travel time is a non-linear function in prevailing the wind conditions.

Since both fuel and time costs are defined arc wise, we aggregate both functions into a single arc-based and time-dependent cost function $c^A \colon A \times \mathbb{R}_0^+ \to \mathbb{R}_0^+$, defined as

$$\begin{aligned} c^A \colon A \times \mathbb{R}_0^+ &\to \mathbb{R}_0^+ \\ (a, \tau) &\mapsto c_f(a, \tau) + c_t(a, \tau). \end{aligned}$$

For the definition of crossing costs for regions, we will closely follow the notation presented by Blanco et al. in [Bla+16b]. We write $\delta^+(v)$ for out-arcs and $\delta^-(v)$ for in-arcs of the node v, and define $\delta(v) := \delta^+(v) \sqcup \delta^-(v)$. We assume that the arcs are partitioned into a set $\mathcal{R}$ of k regions

$$A = R_1 \sqcup R_2 \sqcup \ldots \sqcup R_k,$$

and we call $v \in V$ an **inner node** of R_i if $a \in R_i$ for all $a \in \delta(v)$. Stretching notational limits, we will also write $v \in R_i$ for an inner node v of R_i.

If, conversely, $a \notin R_i$ for all $a \in \delta(v)$, we call v an outer node. All nodes which are neither inner nor outer nodes are called **boundary nodes**, and we write $v \in \partial R$. We count airport nodes as boundary nodes. We emphasise that regions must not overlap arc-wise, yet they may share a common boundary. Without loss of generality, arcs do not cross more than one region: if an arc $a \in A$ did, we could subdivide it and insert a new boundary node at the border.

Write $t(P), h(P)$ for the first (or *tail*) and last (or *head*) node of a path P, and let $\mathcal{S}_R(P)$ denote the set of arc-maximal sub-paths of P in a region $R \in \mathcal{R}$. Then, for a sub-path $p \in \mathcal{S}_R(P)$, its tail $t(p)$ and head $h(p)$ are both elements of ∂R. We will denote the union of all boundary nodes by

$$\begin{aligned} \mathcal{B} &:= \{b \in V : b \in \partial R \text{ for some } R \in \mathcal{R}\} \cup \{v \in V : v \text{ is an airport}\} \\ &= \bigcup_{R \in \mathcal{R}} \partial R \cup \{v \in V : v \text{ is an airport}\}. \end{aligned}$$

Assume a metric $d\colon V \times V \to \mathbb{R}_0^+$. In our application, the natural metric arising from embedding $G = (V, A)$ on a spherical earth model is the great circle distance (gcd). We write $\mathcal{P}_s^t$ for the set of all s-t paths. For a non-decreasing function $f_R\colon \mathbb{R}_0^+ \to \mathbb{R}_0^+$ we can now define the **crossing costs** $c_o^R\colon \mathcal{P}_s^t \to \mathbb{R}_0^+$ for a region R and an s-t-path P as

$$c_o^R(P) := \begin{cases} f_R\left(\sum_{p \in \mathcal{S}_R(P)} d\big(t(p), h(p)\big) \right) & \text{if } R \cap P \neq \emptyset, \\ 0 & \text{if } R \cap P = \emptyset. \end{cases}$$

Note that these costs do not rely on the time τ_0 at all. We can now define the Time-Dependent Shortest Path Problem with Crossing Costs (for short: T-SPROC) as follows:

Input: A directed graph $G = (V, A)$, nodes $s, t \in V$, a departure time $\tau_0 \in \mathbb{R}_0^+$, an arc-based cost function $c^A\colon A \times \mathbb{R}_0^+ \to \mathbb{R}_0^+$, and a crossing cost function $c_o\colon \mathcal{P}_s^t \to \mathbb{R}_0^+$ as defined above.

Objective: Find an s-t-path P starting at τ_0 which minimises

$$c(P, \tau_0) := \sum_{i=0}^{n-1} c^A\big((v_i, v_{i+1}), \tau_i\big) + \sum_{R \in \mathcal{R}} c_o^R(P). \tag{3.1}$$

Remark 3.3. T-SPROC is an extension of TDSPP. Clearly, the first sum constitutes the time-dependency, whereas the second one accounts for the crossing costs. Especially, if $c_o^R(\cdot) \equiv 0$, we re-obtain the TDSPP. Also, note that while the crossing costs c_o are not time-dependent, they are not defined per arc. This obvious difference to TDSPP poses an additional challenge.

We have shown in [Sch16b] that under certain conditions, the wind functions in our application satisfy the FIFO property; we will for the remainder of this paper presuppose the FIFO property for c^A. Blanco et al. developed the Two-Layer-Dijkstra algorithm in [Bla+16b], which solves the shortest path problem with crossing costs to optimality in polynomial time.

We also observe that Theorem 3.1 requires an underestimation function for all nodes $v \in V$. Since we cannot even evaluate the crossing costs for non-boundary nodes $v \in V \setminus \mathcal{B}$, we cannot apply Theorem 3.1 to T-SPROC directly. However, since overflight charges are always non-negative, we can underestimate the function c_o by zero.

3.4 Preprocessing in Practice

In order to have as low runtimes as possible, we aim to prune the a-priori search space for the Flight Planning Problem. Regardless of which pruning algorithm we choose, the objective function will always be given by (3.1).

3.4.1 Dead-End elimination

We can pre-eliminate any nodes which either cannot reach t or cannot be reached from s. This can be done in $\mathcal{O}(|V| + |A|)$, as it suffices to run one forward breadth-first search from s and one backward breadth-first search from t. This pruning method is plainly graph-theoretical, and removes cul-de-sac nodes from the graph. We will compare all other pruning methods to this baseline both in terms of runtime and in terms of nodes in the search space.

3.4.2 The Tank-Capacity Pruning

This pruning method is the most intuitive one. Aircraft clearly cannot burn more fuel than they can carry with them; since fuel burn is proportional to the flight time, this means that there is an inherent maximum flight time for aircraft based on their tank capacity. Let $\overline{\Phi}$ denote the maximum fuel which the aircraft can carry, and $\varphi_s^t(v)$ the fuel consumption on a shortest path from s to t via v.

To underestimate $\varphi_s^t(v)$ for a node $v \in V$, we use the Super-Optimal Wind as introduced by Blanco et al. in [Bla+16a]. The Super-Optimal Wind for an arc $a \in A$ is an artificial wind vector which never underestimates the travel time for an arc $a \in A$ arising out of an actual wind conditions. Given $a \in A$, it is obtained by separately minimising the cross wind and maximising the track wind, leading to the artificial wind vector

$$\mathbf{w}(a) = (\underline{w}_c(a), \overline{w}_t(a)),$$

where $\underline{w}_c(a) := \min_{\tau \in \mathbb{R}_0^+} |w_c(a, \tau)|$ and $\overline{w}_t(a) := \max_{\tau \in \mathbb{R}_0^+} w_t(a, \tau)$. Note that one can obtain better lower bounds by computing the Super-Optimal Wind $\mathbf{w}_i(a)$ per $\tau_i, \tau_{i+1} \in \Delta$. For a full discussion of the topic, we point the reader to [Bla+16a]. As stated in the same work, Super-Optimal Wind can be pre-computed independent of the instance in a few seconds for all arcs $a \in A$.

We create a lower-bound graph $(G, \underline{l})$, by computing the Super-Optimal Wind vector for each arc $a \in A$. This wind corresponds to a **minimum air distance** $\underline{l}_i(a)$ for each arc $a \in A$ and a given wind prognosis time $\tau_i \in \Delta$. Given an upper bound on the travel time (e.g., through a previously computed solution), we can determine $\tau_{k_0}, \tau_{k_j} \in \Delta$ such that the entire flight is in $[\tau_{k_0}, \tau_{k_j}]$. Hence, setting

$$\underline{l}(a) := \min_{i \in \{k_0, \dots, k_j\}} \underline{l}_i(a)$$

provides an effective lower bound on the air distance for the entire flight. We then run a one-to-all-Dijkstra from s and an all-to-one-Dijkstra to t on $(G, \underline{l})$. Then, for any node $v \in V$, we obtain the minimum distance $\underline{l}_s^t(v)$ from s to t via v by setting

$$\underline{l}_s^t(v) := \underline{l}(s, v) + \underline{l}(v, t),$$

where $\underline{l}(u, w)$ denotes the length of a shortest path from u to w in G, with respect to $\underline{l}$. This value is a lower bound on the wind-corrected distance between s and t. We convert $\underline{l}_s^t(v)$ to fuel consumption by assuming an optimal flight profile on the given air distance, to obtain a lower bound $\underline{\varphi}_s^t(v)$ on the fuel consumption via v. Whenever $\underline{\varphi}_s^t(v)$ exceeds the tank capacity $\overline{\Phi}$, we can eliminate v from the search space. Note that while this is similar to Theorem 3.1, we do not rely on a precomputed upper bound; rather, the tank capacity is implicit in the input data. Since all arc costs are static, we can compute this lower bound for every node in $\mathcal{O}(|A| + |V| \log |V|)$.

3.4.3 Fuel and Time Pruning

Since both fuel and time costs are defined arc-wise, it makes sense to use both to create an arc-based underestimation – we will in this subsection underestimate crossing costs by zero.

Underestimations for time costs are easier to obtain than for fuel costs. We again make use of the Super-Optimal Wind: by employing the same strategy as above, we can obtain a lower bound $\underline{T}_s^t(v)$ on the travel time between s and t via any node v. As it turns out, both computations can be done in a single step by using that air distance and travel time are proportional via the constant air speed. Since we assume time costs $c_t(P)$ for a path P to be a linear function in the travel time, they are very easy to underestimate: in fact, the costs for the underestimated travel time are a good underestimation of the actual time costs. We define

$$\underline{c}_t(v) := c_t\big(\underline{T}(s,v) + \underline{T}(v,t)\big)$$

Hence, given an upper bound solution with cost $\overline{c}$, the pruning inequality given in Theorem 3.1 evaluates to

$$\underline{c}_f(v) + \underline{c}_t(v) =: \underline{c}^A(v) \leq \overline{c}, \tag{3.2}$$

and we can eliminate any node $v \in V$ which violates it. This is essentially the application of Theorem 3.1 to the time-dependent part of T-SPROC, with crossing costs underestimated by the constant zero function. As in the tank capacity case, all data is static. In particular, we can compute this lower bound at the cost of Dijkstras, namely in $\mathcal{O}(|A| + |V| \log |V|)$.

3.4.4 Pruning Crossing Costs

The problem with the method presented in the previous section is that although we use the upper bound cost comprising crossing costs, we only sensibly underestimate the fuel and time components (i.e., the time-dependent part of T-SPROC). The crossing costs are underestimated by zero. While still a valid underestimation, overflight charges may account for up to one fifth of the total route costs, depending on the aircraft type. This already justifies the incorporation of these charges into our underestimation.

Overflight charges were already investigated in [Bla+17a], where the authors project crossing costs for regions on the arcs using a heuristic which works very well in practice, but cannot guarantee an underestimation. Instead, we are going to pursue an exact solution. In [Bla+16b], Blanco et al. introduce a macro graph which they use to keep track of the overflight costs. We will mimic this construction, and use the macro graph to underestimate the overflight costs.

So far, all cost components could be computed as a sum of individual arc costs. Recall that as per the definition, crossing costs are not defined per arc, but are only given at the boundaries of regions. Hence, sensible pruning can only occur at these boundaries. The idea is to eliminate as many boundary nodes as possible, and then prune the search space further by running an additional fuel/time pruning on the reduced search space.

Since crossing costs are not time-dependent, we can precompute a lower bound on them by constructing a new graph $M = (V', A')$. We define this **macro graph** M as in [Bla+16b]:

$$V' := \{v \in V : v \in \partial R \cap \partial R' \text{ for some } R, R' \in \mathcal{R}\} \cup \{v \in V : v \text{ is an airport}\},$$

i.e., the new nodes are all boundary nodes of the regions. We also count airport nodes as boundary nodes, to allow for crossing costs for flights beginning or ending in the interior of a region. While instead of all airports it would suffice to only add s and t (as done in [Bla+16b]), our more general definition has the advantage of being independent of the instance. We can hence reuse the same graph structure for all flight instances.[a] We then set

$$A' := \{a = (u, v) \in 2^{V'} : \exists P = (u, n_1, \ldots, n_{k-1}, v) \text{ such that } u, v, n_i \in R \, \forall i\}.$$

In other words, we insert an arc between two boundary nodes of R whenever there is a path connecting them which is entirely contained in R. We then endow M with the (not time-dependent) metric function d as defined in section 3.3. The macro graph for a set of airspaces is depicted in Figure 3.1. Note that for all boundary nodes $b \in \mathcal{B}$, the value

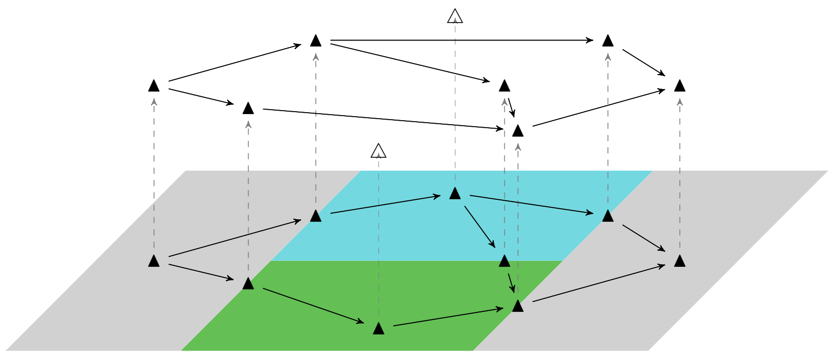


Figure 3.1: Macro graph with two airports in the gray regions.

$$\underline{c}_o(b) := c_o(s, b) + c_o(b, t)$$

is not only a lower bound on the crossing costs, but the actual crossing costs $c_o(b)$ via b. In particular, by running a one-to-all Dijkstra from s and an all-to-one Dijkstra to t, we can obtain the actual crossing costs from s to t via each $b \in \mathcal{B}$. By construction, $|A'| \in \mathcal{O}(|V'|^2)$, which means that running the Dijkstra algorithms takes at most $\mathcal{O}(|V'|^2)$.

We observe that a route which minimises the crossing costs need not be optimal in terms of the total costs – the minimum crossing costs c_o^* between s and t are always a lower bound on the actual crossing costs $c_o(P)$ for any s-t-path P. This means that we can safely underestimate the crossing costs by c_o^* instead of zero, without losing optimality in the ensuing query – thus raising the lower bound by a significant amount. This leads to the following procedure:

1. Deactivate all boundary nodes $b \in \mathcal{B}$ for which $\bar{c} < \underline{c}^A(b) + \underline{c}_o(b)$. Just as for bidirectional Dijkstra[Bas+15a], observe that whenever the fringes of both the one-to-all tree and the all-to-one tree meet at a node b, we obtain a candidate $c_o(b)$ for the minimum overflight costs between s and t. The total minimum overflight costs

$$c_o^* = \min_{b \in \mathcal{B}} c_o(b)$$

are therefore a natural byproduct of this step.

[a]While our definition is very similar to the one in [Bla+16b], the authors use it to solve the shortest path problem with crossing costs to optimality, whereas we use it to prune the search space.

Table 3.1: Pruning methods

DEAD END	dead end elimination
TANK CAP	tank capacity pruning
FUEL TIME	fuel/time pruning
FUEL TIME OFC	fuel/time/overflight costs pruning

2. Lower $\overline{c}$ to $\overline{c}' := \overline{c} - c_o^*$ (this is equivalent to raising the lower bound by c_o^*).
3. Run fuel/time pruning on the reduced search space with the upper bound $\overline{c}'$.

Through this procedure, we use the influence of overflight charges twice: first in actively removing boundary nodes, second by lowering the upper bound for the ensuing fuel/time pruning.

3.5 Computational Results

We implemented all of the pruning algorithms explained in Section 3.4 within the framework of our application. As is the case in almost every application, in ours too we have to consider several extensions, most notable of which are distinct flight levels. To overcome this difficulty, we only consider just one layer of the resulting 3D graph, and represent the other layers only implicitly. We do so by projecting all arcs with the same tail and head nodes (possibly on different levels) onto one representative arc; its underestimated length is then the minimum lower bound length over all levels. Running the pruning algorithms on this 2D representation of the whole graph does not affect the validity of the described methods.

We will measure the quality of the pruning methods by comparing runtimes and the number of active nodes before pruning to afterwards both absolutely and relative to dead-end elimination, which will act as our baseline. A higher number always indicates a more effective method. The advantage of counting active nodes is that the speedup is purely algorithm- but not implementation-dependent. For a fixed instance, we will keep the same upper bound solution for each underestimator for all pruning algorithms. The names for the different pruning methods found in this section are listed in Table 3.1.

Both the Airway Network and the instance data were provided by Lufthansa Systems. The graph consists of 109 314 nodes and 838 114 arcs per level, on 43 such flight levels. The instance set consists of the 7 735 most often flown international connections, based on week 21 of 2014.[b] We limit ourselves to international relations only, since for some countries such as the US, overflight charges do not apply for domestic flights.

All considered flights connect cities which are at least 1 000 km apart on the great circle between the two cities; due to the structure of the Airway Network, instances with airports closer to each other do not benefit much from nuances in pruning algorithms.

We implemented all algorithms in `C++`, compiled with `GCC`, and all our tests were carried out on machines with 132GB of RAM, and an Intel(R) Xeon(R) CPU E5-2690 v4 processor with 2.60GHz and 35.8MB of cache. All queries were run in single-thread mode.

We investigate three different scenarios: we use two different weather prognoses, six months apart (`wth-16` and `wth-17`), and in a third case investigate the situation where there is no wind at all. This `gcd` case highlights the potential for FUEL TIME OFC pruning while eliminating unpredictable effects introduced by the wind. The absolute runtimes for each scenario, averaged over all 7 735 connections, are presented in Table 3.2.

[b]single trip – only one direction is represented in the instance set.

Table 3.2: Average Query Times per Weather

Weather	DEAD END runtime (s)	DEAD END nodes (#)	TANK CAP runtime (s)	TANK CAP nodes (#)	FUEL TIME runtime (s)	FUEL TIME nodes (#)	FUEL TIME OFC runtime (s)	FUEL TIME OFC nodes (#)
wth-16	44.01	31 115	33.70	9 213	15.05	4 125	10.38	3 042
wth-17	44.61	31 120	33.61	9 151	14.89	4 057	10.11	2 963
gcd	17.42	31 123	12.81	8 673	5.50	3 541	3.41	2 391

Recall that we investigate the base problem in flight planning, that of finding a 3D trajectory. Note, too, that we may have to recompute a solution given new restrictions imposed by ANSPs. With this in mind, it is imperative that each instance can be computed as fast as possible. To this end, we compare the query time of Dijkstra's Algorithm after applying TANK CAP, FUEL TIME, and FUEL TIME OFC pruning to the query time after using only DEAD END. For each of the more than 7 000 flights, the speedup is recorded both absolutely and relatively, averaged over the instances, and then summarised for the current weather situation. Note that this approach is possible since our runtimes are in the order of seconds rather than milliseconds, which yields comparatively stable runtime measurements. The previous table already indicates the results, but we also visualise them in Figure 3.2.

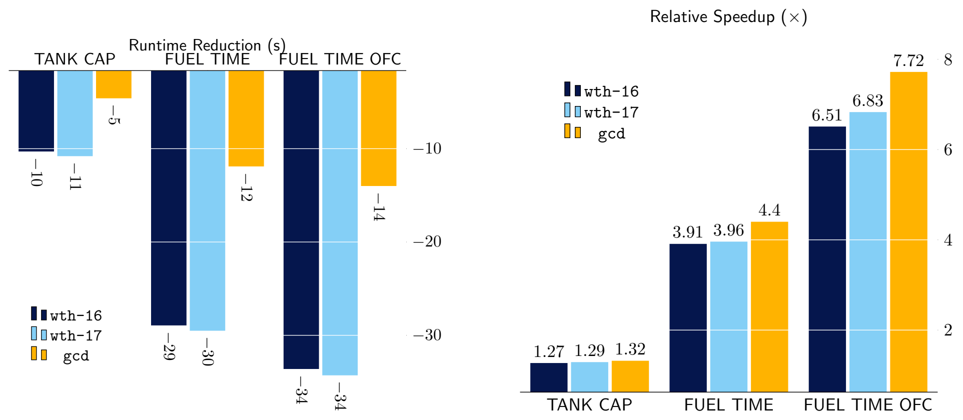


Figure 3.2: Absolute runtime reduction (left) and averaged relative speedup per instance (right), visualised per test set.

One can see that TANK CAP pruning yields a speedup factor close to 1.3, FUEL TIME pruning a factor of almost 4, and FUEL TIME OFC pruning a relative speedup of just under 7 for the weather-dependent instance sets.

To highlight the quality of the respective pruning algorithms, we also recorded the absolute and relative reduction of nodes in the search space, which is only dependent on the pruning method but not on its implementation. The results are presented in Table 3.3. Recall that even though our computations took place in the full 3D setting, our pruning methods work on the projection of the graph onto a 2D layer. Hence, it makes sense measure the reduction of the search space in terms of deactivated 2D nodes.

As one would expect, FUEL TIME OFC pruning is the most effective method. Applying FUEL TIME OFC pruning to the search space yields a reduction of more than 90% of the active nodes. This reduction is also visible in terms of the average query time speedup, which is even ≈ 1.72

Table 3.3: Average 2D Search Space Reduction Per Instance

	TANK CAP		FUEL TIME		FUEL TIME OFC	
Weather	absolute	%	absolute	%	absolute	%
wth-16	21 900	70.84	26 988	87.40	28 071	90.94
wth-17	21 969	71.03	27 063	87.60	28 157	91.18
gcd	22 448	72.45	27 581	89.11	28 731	92.85

times better than with the second best method, FUEL TIME pruning. While the TANK CAP pruning is not quite as effective as the other methods, its inherent advantage is that it does not rely on an upper bound solution. Indeed, the upper bound is given by the input in terms of the aircraft's tank capacity, which renders it a computationally light alternative – or a fallback method in case one cannot find a reasonable upper bound solution.

It becomes apparent that all pruning methods are more effective in the gcd case than with weather. This is logical since this case can be thought of as zero wind, whose Super-Optimal Wind underestimation is perfect – thus tightening the bounds on both fuel and time underestimation. Consequently, the influence of including the overflight fees in the estimation become more apparent. Therefore, it is also not surprising that FUEL TIME OFC pruning in the gcd case is the most effective of all methods, since it deals with the tightest lower and upper bounds possible.

3.6 Conclusion

We have investigated the Flight Planning Problem in more detail than what was covered before. To speed up the query, we have developed and presented three different pruning methods for an a priori search space reduction. In particular, we presented a way to incorporate crossing costs in the underestimation, thus tightening the lower bounds on the optimal costs for the Flight Planning Problem.

We showed both theoretically and computationally that each of the methods is effective. Clearly, including crossing costs in the underestimation yields a noticeable reduction of both the search space and the ensuing query time. While all pruning methods bear similarities to A*, to the best of our knowledge, there is no known underestimator for this particular problem. This is partly due to the fact that A* requires that one define a potential function at each node. Contrastingly, we use a two-stage pruning process to first eliminate boundary nodes and then deactivate inner nodes.

The added benefit of a priori search space reduction is that it can be used with non-heap-based algorithms, such as topological sorting, too.

References

[Bas+15a] Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. *Route Planning in Transportation Networks.* Tech. rep. Microsoft Research, 2015.

[Bat+13] G. Veit Batz, Robert Geisberger, Peter Sanders, and Christian Vetter. "Minimum Time-dependent Travel Times with Contraction Hierarchies". In: *Journal of Experimental Algorithmics* 18 (Apr. 2013), 14:11–14:143. ISSN: 1084-6654.

[Bau+14] Moritz Baum, Julian Dibbelt, L. Hübschle-Schneider, T. Pajor, and D. Wagner. "Speed-consumption tradeoff for electric vehicle route planning". In: *OpenAccess Series in Informatics* 42 (Sept. 2014), pp. 138–151. DOI: `10.4230/OASIcs.ATMOS.2014.138`.

[Bla+16a] Marco Blanco, Ralf Borndörfer, Nam Dung Hoang, Anton Kaier, Adam Schienle, Thomas Schlechte, and Swen Schlobach. "Solving Time Dependent Shortest Path Problems on Airway Networks Using Super-Optimal Wind". In: *16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2016).* 2016.

[Bla+16b] Marco Blanco, Ralf Borndörfer, Nam Dung Hoang, Anton Kaier, Thomas Schlechte, and Swen Schlobach. *The Shortest Path Problem with Crossing Costs.* Tech. rep. ZIB, 2016.

[Bla+17a] Marco Blanco, Ralf Borndörfer, Nam Dung Hoang, Anton Kaier, Pedro Maristany de las Casas, Thomas Schlechte, and Swen Schlobach. "Cost Projection Methods for the Shortest Path Problem with Crossing Costs". In: *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017).* 2017.

[DG17] Axel Dreves and Matthias Gerdts. *Free Flight Trajectory Optimization and Generalized Nash Equilibria in Conflicting Situations.* Tech. rep. Universität der Bundeswehr München, 2017.

[Dij59] Edsger W. Dijkstra. "Numerische Mathematik". In: Springer, 1959. Chap. A Note on Two Problems in Connexion with Graphs, pp. 269–271.

[Dre69] Stuart E. Dreyfus. "An Appraisal of Some Shortest Path Algorithm". In: *Operational Research* 17.3 (June 1969), pp. 395–412.

[Eur19a] Eurocontrol. *Free route airspace (FRA).* English. Accessed April 2019. EUROCONTROL. 2019. URL: `https://www.eurocontrol.int/articles/free-route-airspace`.

[Eur19b] Eurocontrol. *Route Availability Document.* Eurocontrol. Apr. 2019. URL: `https://www.nm.eurocontrol.int/RAD/`.

[FHS14] Luca Foschini, John Hershberger, and Subhash Suri. "On the Complexity of Time-Dependent Shortest Paths". In: *Algorithmica* 68.4 (Apr. 2014), pp. 1075–1097. ISSN: 1432-0541. DOI: `10.1007/s00453-012-9714-7`. URL: `https://doi.org/10.1007/s00453-012-9714-7`.

[FT87] Michael L. Fredman and Robert Endre Tarjan. "Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms". In: *Journal of the ACM* 34.3 (July 1987), pp. 596–615.

[Gei+12] Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. "Exact Routing in Large Road Networks Using Contraction Hierarchies". In: *Transportation Science* 46.3 (Aug. 2012), pp. 388–404. ISSN: 1526-5447.

[HNR68b] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *IEEE Transactions of Systems Science and Cybernetics* 4.2 (July 1968), pp. 100–107.

[JCL17a] Casper Kehlet Jensen, Marco Chiarandini, and Kim S. Larsen. "Flight Planning in Free Route Airspaces". In: *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017).* 2017.

[Kar+12b] Stefan E. Karisch, Stephen S. Altus, Goran Stojković, and Mirela Stojković. “Quantitative Problem Solving Methods in the Airline Industry”. In: ed. by Cindy Barnhart and Barry Smith. Springer, 2012. Chap. Operations, pp. 283–383.

[Sch16b] Adam Schienle. “Shortest Paths on Airway Networks”. Master’s Thesis. Freie Universität Berlin, 2016.

Introduction to Chapter 4

This chapter was published in [BBC22].

We present an A* algorithm running on the three-dimensional space and optimizing fuel consumption under consideration of wind. It relies on the travel time underestimators introduced in Chapter 2. We also generalize the fuel underestimation technique used (albeit not so well described) in Chapter 3.

As mentioned in Chapter 1, there is a lack of serious academic research on the Flight Planning Problem that is still relevant in the modern day. A consequence of this is that it is difficult or impossible to find benchmarks against which to compare new algorithms. The publication of [KCL18] provided us with an excellent work against which to test our methods. It presents an A* algorithm whose potential function uses the *Single Descent* (SD) technique to underestimate fuel consumption from any search label to the destination.

Like that work, we introduce an A* algorithm that runs on a three-dimensional graph and takes both weather and fuel consumption into account. While [KCL18] notes that its best performing potential function is neither admissible nor consistent, leading to an algorithm without optimality guarantees, we have been able to define reasonable conditions under which ours satisfies both properties.

Furthermore, our algorithm has a better performance on real-world data, thanks in part to our preprocessing techniques. We ensure that the comparison is fair by re-implementing the calculation of the SD potential function. The remaining parts of the implementation (label management, priority queue, etc.) are exactly the same for both algorithms. Note that while [KCL18] considers traffic restrictions and we do not, the SD potential is defined independently of them. Our potential function could be easily extended to handle restrictions by using similar methods as [KCL18], which utilizes the multi-label technique first introduced in [KCL17]. It is reasonable to anticipate that the performance gain in comparison to Single Descent (in the restriction-free case considered in this chapter) would be maintained even with the inclusion of flight restrictions. The same applies to the pruning heuristics presented in that paper.

An algorithm using our *Idealized Vertical Profiles* potential function combined with the restriction handling from [KCL17] would thus be, to the best of our knowledge and as of the publication time of this thesis, the fastest known algorithm for solving the Flight Planning Problem.

References

[BBC22] Marco Blanco, Ralf Borndörfer, and Pedro Maristany de las Casas. "An A* Algorithm for Flight Planning Based on Idealized Vertical Profiles". In: *22nd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems.* Vol. 106. This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by/4.0/. 2022, 1:1–1:15. DOI: https://doi.org/10.4230/OASIcs.ATMOS.2022.1.

[KCL17] Anders Nicolai Knudsen, Marco Chiarandini, and Kim S. Larsen. "Constraint Handling in Flight Planning". In: *Principles and Practice of Constraint Programming - 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*. 2017, pp. 354–369. DOI: 10.1007/978-3-319-66158-2_23. URL: https://doi.org/10.1007/978-3-319-66158-2_23.

[KCL18] Anders Nicolai Knudsen, Marco Chiarandini, and Kim S. Larsen. "Heuristic Variants of A* Search for 3D Flight Planning". In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 15th International Conference, CPAIOR 2018, Delft, The Netherlands, June 26-29, 2018, Proceedings*. 2018, pp. 361–376. DOI: 10.1007/978-3-319-93031-2_26. URL: https://doi.org/10.1007/978-3-319-93031-2_26.

Chapter 4

An A* Algorithm for Flight Planning based on Idealized Vertical Profiles

Marco Blanco, Ralf Borndörfer, Pedro Maristany de las Casas

Abstract

The Flight Planning Problem is to find a minimum fuel trajectory between two airports in a 3D airway network under consideration of the wind. We show that this problem is NP-hard, even in its most basic version. We then present a novel A* heuristic, whose potential function is derived from an idealized vertical profile over the remaining flight distance. This potential is, under rather general assumptions, both admissible and consistent and it can be computed efficiently. The method outperforms the state-of-the-art heuristic on real-life instances.

4.1 Introduction

The Flight Planning Problem (FPP) seeks to compute a flight trajectory between two airports that minimizes fuel consumption. In this paper we consider a basic version subject to weather conditions, aircraft performance, and an airway network.

Weather forecasts for flight planning are usually provided on a 4D grid, which specifies a wind vector for each coordinate, altitude, and time. These data can be interpolated on all 4 dimensions to obtain a single wind vector acting on each flight segment, see [Bla+16d] for more details. For the purposes of this paper, it suffices to think of wind as a function that maps time to an effective *air distance* that is needed to traverse a given segment.

Aircraft performance specifies how the state of the aircraft changes as a function of the flight phase and various parameters. Namely, for the current weight, the current altitude, the target distance, and the local wind condition, the performance function computes the weight after a cruise, climb, or descent phase along a flight segment. For cruise phases, the influence of the wind can be subsumed into the distance to the cruise target. The fuel consumption is then the weight difference, while the cruise time can be easily calculated from the speed (which we assume here to be constant) and the distance. For climbs and descents, distance, consumption, and time are more difficult to compute, since they depend on the vertical angle, which in turn

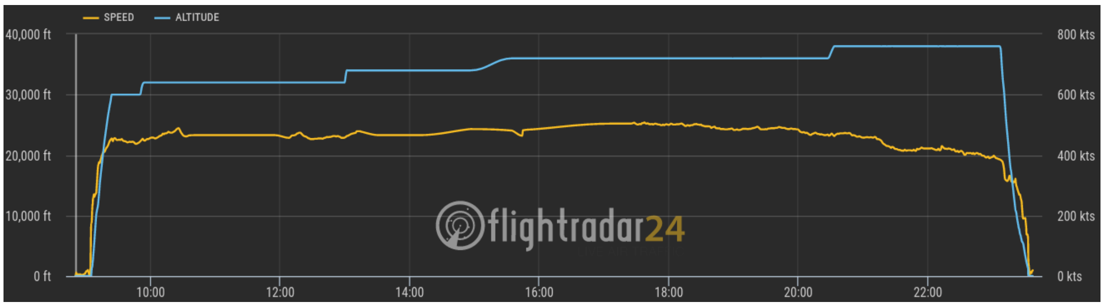


Figure 4.1: Vertical profile on a flight from Amsterdam to Santiago. The blue graph represents the aircraft's altitude over time. Image obtained from FlightRadar24.com on the 9th of July 2021.

depends on the aircraft weight. In accordance with the literature, see, e.g., [KCL18], we assume that, ceteris paribus, a higher weight results in a higher consumption, that cruising is in general more efficient as the altitude increases, until an *optimal cruise altitude* is reached, and that a smaller aircraft weight results in a steeper climb/descent angle. Moreover, a climb between two given altitudes might not be possible if the aircraft weight exceeds a certain threshold. These properties produce a vertical profile shape that is known as *step-climb*. Namely, to fly efficiently, an aircraft climbs from the departure airport to the highest altitude reachable in a single climb. Then, it cruises on this altitude until it has burned enough fuel and is light enough to climb further. This is repeated until the optimal cruise level is reached. Finally, the aircraft needs to start the final descent. See Figure 4.1 for a real-life example.

The *Airway Network* is a directed graph with a three-dimensional embedding covering the airspace around the Earth. It arises from a set of waypoints (2D coordinates) connected by airway segments (straight lines) on a set of discrete flight levels (altitudes); airports are a modeled as a particular type of waypoints. The horizontal profile of a legal flight trajectory must consist of a contiguous sequence of airway segments connecting the two airports. Vertically, cruise phases are only allowed on one of the flight levels, while a climb or descent phase must be started at a waypoint (it cannot be started from the interior of a segment, but can and usually does end in the interior).[a]

The literature on the FPP varies greatly in the extent and depth at which the technical aspects of the problem are treated. [Jon74] is an extensive work that goes into great detail. To the best of our knowledge, it presented the first dynamic programming algorithm which runs on a 3D graph. [HM98] uses a dynamic programming approach to minimize fuel consumption during the cruise phase for a fixed horizontal route. [NSG14] computes a trajectory on a search space where the horizontal route is not restricted by waypoints and segments by splitting the problem into a horizontal and a vertical component, which are solved sequentially using dynamic programming approaches. [Kar+12a] gives a realistic and detailed survey of the most relevant cost components and restrictions, as well as an excellent review of previous work. The authors sketch some possible ways of solving the problem, such as decomposition into horizontal and vertical optimization (2D+2D) or 4D search, all on a high level.

The FPP can be seen as a special route planning problem. In this domain, A* algorithms achieve excellent running times. The main idea of these algorithms is to guide a Dijkstra-

[a]In practice, the final descent to the destination airport is an exception, since it can be started everywhere. In this work, we ignore this exception for the sake of simplicity and without a significant impact on the results.

like search towards the potential function which is built in a preprocessing stage. Potential functions map the nodes of the graph onto estimates that bound the cost of a shortest path to the target node from below. There is extensive literature that focuses on the task of computing such heuristic functions and thus designing A* algorithms for routing on (time dependent) road networks [DN12; Bas+15b; SZ19], routing for electric vehicles [Bau+20], or in multiobjective scenarios [MSB21; Mar+21].

A* algorithms have also been considered for the FPP. A series of papers by a group from the University of Southern Denmark studies the problem in a very realistic way, presents new algorithms, and tests them on real-world data: [KCL16] optimizes the vertical profile of a given 2D-route by a simple A* algorithm whose lower bounds are calculated from the minimum consumption on each arc. It also shows that even though the FIFO property[b] does not hold due to the unpredictable nature of weather, the Dijkstra algorithm in practice nearly always finds an optimal solution. [KCL17] provides an algorithm to solve flight planning under consideration of traffic restrictions, for the case of a constant flight altitude. It is based on storing multiple labels per node/altitude pair. [JCL17b] discusses the *free-route* case, where the flight area is not limited by an Airway Network. The most relevant article for our work is [KCL18]. It considers the same setting as ours plus flight restrictions, which are handled by the algorithm from [KCL18]. The main contribution of the paper are two variants of an A*-type algorithm on a three-dimensional graph, called *All Descents* and *Single Descent.* The first one uses very conservative lower bounds on the arc lengths, which are used for a backwards search that defines the potentials; these are both admissible and consistent under the FIFO assumption. The Single Descent algorithm calculates the potentials partially before the start of the search and partially during the expansion of the labels. It is much faster than the All Descents variant, but the potentials are neither admissible nor consistent. However, the computational results testify a very small error on real-world instances. We will use the Single Descent algorithm as a benchmark in our computations.

This paper builds on our previous work [Bla+16d], which investigates the FPP restricted to a constant altitude. It presents a method for calculating lower bounds on travel-time on arcs by using a concept called *super-optimal wind.* This in turn is used to construct potentials for an A* algorithm.

While the addition of altitudes requires a much more sophisticated approach, the distance underestimation techniques of [Bla+16d] are a critical component of our new algorithm. [Bla+17b] presents a heuristic that handles complex overflight costs by reducing them to classical costs on arcs by solving a Linear Program. This approach can be trivially combined with most others, including the one we present. Finally, [SMB19] also investigates a horizontal variant of the FPP, which considers both weather and overflight costs. It introduces efficient pruning techniques that reduce the graph before the start of the search algorithm. These techniques can also be easily incorporated in a step preceding an A* search.

The FPP is a time-dependent shortest path problem on the Airway Network subject to weather conditions and aircraft performance. We show that it is NP hard, a basic fact that, as far as we know, has hitherto not been noted. As such, the FPP cannot be solved by a Dijkstra-type label setting algorithm. However, as this approach is efficient and produces excellent results, it is commonly used in practice and also as our benchmark in this paper. In this vein, we present an A*-algorithm that improves on Dijkstra's algorithm. Its potential function is the cost of an idealized vertical trajectory over a lower bound of the total remaining flight distance. The construction of this idealized trajectory is based on the above mentioned assumptions about optimal vertical profiles. We show that it can be calculated efficiently on-the-fly, during the label expansion, and further sped-up by a pre-calculation of parts of the climb phase that

[b]The FIFO property states that early arrival is always beneficial.

depends only on the aircraft type, i.e., the aircraft performance function. This leads to a fast algorithm, which is essential in order to account for the latest weather forecast and the newest flight restrictions. On a set of real-world instances, our approach is on average 7-10 times faster than Dijkstra's algorithm and 30-40% faster than the Single Descent algorithm of [KCL18].

The paper is structured as follows. In Section 4.2 we present a mathematical model of the Flight Planning Problem (FPP) that generalizes the Time-Dependent Shortest-Path-Problem (TDSPP). We also present the first NP-hardness proof for the FPP; this proof extends to a large family of TDSPPs. Section 4.3 presents an A*-algorithm for the FPP. Its potential function computes the cost of an idealized vertical profile over a lower bound of the total remaining flight distance. Under certain assumptions on aircraft performance, this potential is admissible and consistent, and it can be computed efficiently. In Section 4.4, we compare our implementations of the new A*-algorithm, Dijkstra's algorithm, and the Single Descent algorithm. The results show that the potential calculation pays off by drastically reducing the number of expanded labels and the runtime. They also show that our consistency assumptions are satisfied to a reasonable degree.

4.2 The Flight Planning Problem

We represent the Airway Network by a directed graph $G = (V, A)$. Each waypoint gives rise to multiple nodes, corresponding to the different flight levels H; denote by $h(v) \in H$ the flight level of node v. We assume that the departure and the arrival airport are located not on the ground but on the lowest flight level h_0[c]. Likewise, each segment gives rise to multiple arcs: One cruise arc for each flight level and one climb or descent arc for each combination of two flight levels. We assume that the highest flight level is the optimal cruise level, since it does not make sense to fly higher. Both aircraft performance functions and wind are handled by a *propagation function* $\boldsymbol{\tau} : W \times T \times A \to (W \cup \{\infty\}) \times T$; here, $W \subset \mathbb{R}$ is a set of weights, ∞ represents an infeasible state, and $T \subset \mathbb{R}$ a set of times. Then, the propagation function maps the state of the aircraft at the tail of an arc to its state after traversing the arc. We assume the following propagation properties.

Assumption 1. *Let $\boldsymbol{\tau} : W \times T \times A \to (W \cup \{\infty\}) \times T$ be a propagation function. For $w_1, w_2 \in W$, $t \in T$, $a_1, a_2 \in A$, $\boldsymbol{\tau}(w_1, t, a) = (w^1, t^1)$, and $\boldsymbol{\tau}(w_2, t, a) = (w^2, t^2)$, it holds:*

i) $w_1 > w^1$ *and* $t < t^1$,

ii) $w_1 < w_2, a_1 = a_2 \Longrightarrow (w_1 - w^1) < (w_2 - w^2)$,

iii) $w_1 = w_2, a_1, a_2$ *cruise arcs with* a_2 *on a higher level* $\Longrightarrow (w_1 - w^1) > (w_2 - w^2)$,

iv) *ceteris paribus, a descent burns less fuel than a cruise, which burns less than a climb, and a direct descent, if possible, is the most economic way to reach the destination.*

v) *For fixed* $a \in A$, $t \in T$, *the* air distance *along a at time t (i.e. the effectively traversed distance, after consideration of wind) is proportional to* $w_1 - w^1$.

i) states that traversing an arc decreases the weight (by burning fuel) and increases time. ii) means that fuel consumption increases with weight. iii) says that fuel consumption on a cruise phase decreases with altitude[d], iv) is clear. v) states that consumption increases with air distance, which is very intuitive. With these definitions, the FPP can be stated as follows.

[c]In our data, this corresponds to an altitude of 300m; the final descent ends on FL h_0.

[d]Recall that we assume that the highest flight level is the optimal one.

Definition 4.1. Let $G = (V, A)$ be an Airway Network and $v^{DEP}, v^{DEST} \in V$ be the nodes corresponding to the departure and destination airports, respectively. Let $t^0 \in T$ and $w^0 \in W$ be the weight and time at departure, and $\boldsymbol{\tau} : W \times T \times A \to W \times T$ a propagation function. The Flight Planning Problem (FPP) seeks to find a path $((v_0, v_1), (v_1, v_2), \dots, (v_{n-1}, v_n)) \subset A, n \in \mathbb{N}$, and corresponding sequences of weights $(w_0, w_1, \dots, w_n) \subset W$ and times $(t_0, t_1, \dots, t_n) \subset T$. It must hold that $v_0 = v^{DEP}$, $v_n = v^{DEST}$, $w_0 = w^0$, $t_0 = t^0$, and $\boldsymbol{\tau}(w_i, t_i, (v_i, v_{i+1}) = (w_{i+1}, t_{i+1})) \in W \times T$ for each arc (v_i, v_{i+1}) in the path. The objective is to minimize $w^0 - w_n$.[e]

While some variants of the FPP investigated in the literature are solvable in polynomial time [Bla+16d] under certain assumptions, others are clearly NP-hard ([KCL17], [Bla+16e]). [KCL16] notes that the FIFO property does not hold under the presence of wind, but that by itself does not have any implications on the computational complexity of the problem.

In this section, we show that the version of the FPP considered in this paper is NP-hard, even without consideration of wind. We first note that the weight parameter in the FPP is equivalent to the time parameter in the classical Time-Dependent Shortest Path Problem (TDSPP), such that we can think of fuel propagation functions as traversal-time functions. It is well known that the FIFO property is a sufficient but not a necessary condition for the TDSPP to be solvable in polynomial time, while [OR89] gave the most widely cited proof that the TDSPP can be NP-hard in non-FIFO networks. They construct travel time functions on a finite domain that have a constant value except for one point. As our fuel propagation functions do not have this structure, [OR89]'s argument cannot be applied. The same holds for the proof in [Zei23]. To the best of our knowledge, no other published proofs would apply to the FPP. We therefore give a new simple NP-hardness proof based on a more general argument.

Consider the situation in Figure 4.2. Essentially, an arc representing a climb can sometimes only be flown if the aircraft's weight is small enough, as otherwise the higher level cannot be reached before the end of the segment represented by the arc. In other words, the consumption given by $\boldsymbol{\tau}$ on this arc is finite for weights up to a certain value and jumps to infinity for weights above that value. The phenomenon in Figure 4.2 is not as contrived as it may seem. To give the reader an idea of the variability of the climb angle: An Airbus A340 with a typical weight of 200t needs roughly 150km of horizontal flight to climb from 5000m altitude to 10000m altitude. Around this weight, an increase of 1kg roughly leads to an increase of 1m in horizontal distance. The FPP is thus a generalization of the TDSPP that allows at most one jump discontinuity in each travel time function, while the proof in [OR89] assumes two.

Theorem 4.1. *If traversal time functions are allowed to have at most one jump discontinuity, the TDSPP is NP-Hard.*

Proof. Inspired by [He+20], we do a reduction from the Exact Path Length (EPL) problem, which is NP-hard according to [NU02]. Consider a directed graph $G = (V, A)$ with non-negative lengths on the arcs $c : A \to [0, \infty)$, two nodes s, t, and $L > 0$. The EPL consists of determining whether an (s, t)-path of length L exists. For the reduction, we define travel time functions on G as follows. Let $M \gg 0$ be a very large number. Without loss of generality we assume that the departure time is 0. For each $a \in A$ and $\tau \in [0, \infty)$, and using $h(a)$ to denote the head of a, we define [f]

$$T(a, \tau) = \begin{cases} M & \text{if } a = (v, t), \tau < L - c(a) \\ c(a) & \text{else.} \end{cases}$$

If the last arc $a = (v, t)$ of any (s, t)-path is entered at time $\tau < L - c(a)$, the objective value will be larger than M. Thus, L is the smallest possible arrival time, and if a path with

[e]Of course, since w^0 is constant, this is equivalent to maximizing w_n.

[f]Note that this step function does not satisfy the FIFO property, despite similar functions in the literature preserving it, such as [EFS11].

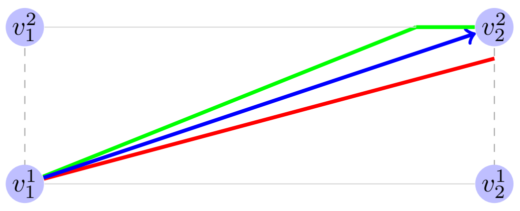


Figure 4.2: The green profile represents a climb along a segment and between two levels. The climb is steep enough that the higher level is reached before the end of the segment, and the aircraft can cruise until reaching node v_2^2. This climb is represented in the graph by the blue arc (v_1^1, v_2^2). The red profile shows a climb started with a higher weight. This leads to a flatter climb, making it impossible to reach v_2^2. Thus, when starting with this weight, the blue arc has an infinite consumption.

this arrival time exists, the TDSPP will find it. Clearly, any path with travel time L also has length L, and vice-versa. Consequently, feasible solutions of the EPL problem correspond to optimal solutions for the TDSPP constructed above. This completes the reduction. □

4.3 An A* Algorithm based on Idealized Vertical Profiles

4.3.1 The state of the art

Recall that an A* algorithm is based on a potential function $\pi : V \to \mathbb{R} \cup \{\infty\}$. π is said to be *admissible* if $\pi(v)$ is a lower bound on the costs from v to the target for $v \in V$. It is *consistent* if $\pi(u) - \pi(v) \leq c(u, v)$ for $(u, v) \in A$, where c is the cost function. Given that $\pi(t) = 0$ for the target t, which we can and henceforth will assume w.l.o.g., consistency implies admissibility, and a consistent and admissible potential guarantees that an A* algorithm finds the same solution as Dijkstra's algorithm. Tight, consistent potentials lead to a faster A* algorithm. The Single Descent (SD) algorithm of [KCL18] pursues such an idea for the FPP: For each segment, a lower bound on the fuel consumption over it is computed as a cruise on the optimal flight level, with the optimal wind conditions, and the minimum possible aircraft weight. On a 2D projection, a backwards Dijkstra search from the destination airport is done w.r.t. these arc costs. The resulting distances are used as initial potentials. In the forward search during the expansion of each label, a descent from that label to the ground is calculated and the corresponding consumption is added to the initial potential. Since the distance traversed by that descent is now covered both by a cruise (initial potential) and a descent (first correction), a second and final correction step is made: The distance of that descent is traversed from the label in cruise mode, and this consumption is subtracted from the preliminary potential, thus defining the final potential.[g] Despite the resulting potentials being neither admissible nor consistent, the ensuing, label-setting, SD algorithm is very effective in practice and marks the current state-of-the-art. Our motivation for improvement is that the initial potentials can be very loose for labels that are far away from the destination, since SD assumes a very low weight and no climbs. This can lead to a significant underestimation of the consumption. The on-the fly correction in the forward search mitigates this problem, but does not resolve it completely.

[g]This is how we interpret the algorithm. Unfortunately, the paper is not very detailed, in particular, w.r.t. the construction of the lower bounds.

4.3.2 Basic framework

We propose an A* algorithm based on a simple idea[h]. In practice, flight routes are constrained by the airway network and by wind conditions. If neither of these existed, but cruise phases were still constrained to flight levels, and the routes would always follow the step-climb pattern, see again Figure 4.1. Namely, the horizontal route component would be the great-circle line connecting both airports, while the vertical route component would consist of a series of climb-cruise-climb sequences up to the optimal flight level. There, the aircraft would cruise until the final descent is started, which would take it straight to the destination airport. The consumption arising from this *idealized vertical profile* (IVP) on a lower bound on the flight distance is an admissible (and, as it will turn out, also consistent) potential for an A* algorithm. Calculating the IVP during the search is too costly, as the decrease in the number of labels would be offset by the effort to compute the potentials. However, it will turn out that this problem can be overcome by a combination of preprocessing and on-the-fly calculations. A formal description is as follows.

A crucial element is the distance underestimation. The results in this section are of a general nature and the specific type of underestimation is not important. In our implementation, we will obtain using the technique introduced in [Bla+16d] ("super-optimal" wind calculations combined with a backwards search).

Definition 4.2. Let $c^{\text{IVP}}(w, h, d, h_T)$ be the minimum amount of fuel that is needed to fly, assuming no wind influence, the distance d by doing some combination of climb/cruise/descent phases, starting at altitude h with weight w, and finishing at altitude h_T; if the distance is too short to reach the target altitude, it is the amount of fuel that is needed to make an immediate descent to the target altitude; if the target cannot be reached, it is infinity. In the first two cases, the vertical profile $p^{\text{IVP}}(w, h, d, h_T)$ of the associated trajectory is called *idealized vertical profile* (IVP).

Assumption 2. *Every IVP consists of a finite and alternating series of climb and cruise phases followed by a single descent phase.*

Assumption 3. $c^{IVP}(w, h, d_1, h_T) \leq c^{IVP}(w, h, d_2, h_T)$ *for distances* $d_1 \leq d_2$.

In other words, Assumption 2 means that the IVP looks like the one in Figure 4.1. It also implies that the highest level reached by the IVP is at most the aircraft's optimal cruise altitude.[i] Assumption 3 means that, ceteris paribus, longer trajectories are more expensive.

Proposition 4.1. *For an FPP and $v \in V$, consider a (v^{DEP}, v)-path in G that reaches v with weight w at time t. Let $h(v)$ be the flight level at v. Let $\underline{d}$ be a lower bound on the distance from v to v^{DEST} obtained by a backwards search from v^{DEST} using lower bounds on the (time-dependent) arc distances as costs; recall that h_0 is the lowest flight level. Then the function*

$$c : V \times W \to [0, \infty], (v, w) \mapsto c^{IVP}(w, h(v), \underline{d}, h_0)$$

is admissible and consistent. [j]

Proof. Admissibility follows from Assumptions 2 and 3, as well as part v) of Assumption 1. Indeed, the air distance traversed by the best possible trajectory following any (v, v^{DEST})-path

[h] The algorithm is label-setting and necessarily heuristic, as the FPP is NP hard.

[i] See Section 4.2 for the definition of the optimal cruise altitude.

[j] We define admissibility and consistency for a function with domain $V \times W$ in the canonically extended way. It is easy to see that all known properties still hold.

in G is at least $\underline{d}$, and its vertical profile is constrained by starting climbs only over waypoints. Hence, it burns more fuel than the IVP $p^{\text{IVP}}(v, w, \underline{d}, h_0)$.

For consistency, consider an arc $(u, v) \in A$, a weight w_u, and a time t_u. Traversing the arc with the initial state (w_u, t_u) leads to the state $\boldsymbol{\tau}(w_u, t_u, (u, v)) = (w_v, t_v)$, so the consumption on the arc is $w_u - w_v$. We must thus prove $c(u, w_u) - c(v, w_v) < w_u - w_v$.

Let $\underline{d}_u$ and $\underline{d}_v$ be the lower bounds used to calculate the potentials $c(u, w_u)$ and $c(v, w_v)$ from IVPs $p^{\text{IVP}}(w_u, h(u), \underline{d}_u, h_0)$ and $p^{\text{IVP}}(w_v, h(v), \underline{d}_v, h_0)$ and let $d(u, v, w_u, t_u)$ be the actual air distance traversed on (u, v). We now distinguish three cases; Case 1 is the standard "en route" case, Cases 2 and 3 come up close to the destination, when the lower bound on the remaining distance becomes small.

Case 1: Neither $p^{\text{IVP}}(w_u, h(u), \underline{d}_u, h_0)$ nor $p(w_v, h(v), \underline{d}_v, h_0)$ are immediate descents. Then

$$\begin{aligned} c(u, w_u) = c^{\text{IVP}}(w_u, h_u, \underline{d}_u, h_0) &\leq c^{\text{IVP}}(w_u, h_u, d(u, v, w_u, t_u) + \underline{d}_v, h_0) \\ &\leq w_u - w_v + c^{\text{IVP}}(w_v, h_v, \underline{d_v}, h_0) \\ &= w_u - w_v + c(v, w_v), \end{aligned}$$

where the first inequality follows from the triangle inequality $\underline{d}_u \leq d(u, v, w_u, t_u) + \underline{d}_v$ for distance lower bounds and Assumption 3, and the second from the optimality of the IVP $p^{\text{IVP}}(w_u, h_u, d(u, v, w_u, t_u) + \underline{d}_v, h_0)$, which burns at most the same amount of fuel as the concatenation of (u, v) and the IVP $p^{\text{IVP}}(w_v, h(v), \underline{d}_v, h_0)$.

Case 2: Only $p^{\text{IVP}}(w_v, h(v), \underline{d}_v, h_0)$ is an immediate descent. The same argument as in Case 1 applies, since the total distance traversed on (u, v) and then on the descent from v will be longer than the distance traversed by the IVP starting at u.

Case 3: $p^{\text{IVP}}(w_u, h(u), \underline{d}_u, h_0)$ is an immediate descent. Now the relative lengths of lower bounds on the traversed air distances are unclear, because a descent is steeper with a lower weight, possibly causing the profile via v to end up with a smaller distance bound. However, we can use Assumption 1 iv) on aircraft performance which implies that a direct descent burns less fuel than any other combination of flight phases leading to the same altitude. $\square$

4.3.3 Calculation of the Idealized Vertical Profile

In the previous section, we proved that the A* potentials calculated using the IVP method are admissible and consistent under consideration of two assumptions. Now, we sketch how these potentials can be computed in practice.

Assumption 2 states that an IVP follows a step-climb procedure until reaching the highest level that allows a direct descent to the ground. Each cruise in this profile is just long enough to burn enough fuel to reach a weight that allows a further climb. Once the highest level is reached, the aircraft cruises until the point where it starts the final descent.

To speed-up the first part of the calculation we observe that the weight at the end of a cruise and at the start of a subsequent climb is constant for a fixed flight level. This is because, by definition, this is the largest weight that allows starting a climb from that level. Similarly, the distances of each such phase are constant. This allows us to pre-compute, for each pair of levels, the total consumption and total distance corresponding to a step climb between these levels, as well as the weight at the start of this step-climb.

The second phase of the calculation, consisting of a single cruise followed by a descent, is trickier. The reason is that the cruise distance plus the descent distance must equal the remaining distance, but the descent distance depends on the weight at its start, which in turn depends on the length of the cruise. In practice, the top of descent is computed by an iterative procedure that progressively adjusts the cruise distance until a total cruise+descent distance is reached that is close enough to the target distance. This procedure can be very time-consuming, which makes it another good candidate for pre-computation. The difficulty is that more parameters are involved than in the step-climb case: Both the weight before the cruise-descent and the remaining distance are unclear.

We solve this problem in the following pre-processing step: For each flight level, we calculate the maximum descent distance from that level to the ground. We then consider a discretization of the complete weight range (that is, from the aircraft's dry operating weight to its maximum take-off weight). For each weight in this discretization, we compute the (time-consuming) IVP on the remaining distance.

The complete calculation of the potentials is described in Algorithm 1 for a weight w, a flight level h, and a remaining distance d. In a nutshell, we compute the IVP as described above. Step-climbs are not calculated on-the-fly, instead we use the pre-calculated data. Near the destination airport, we use the second batch of pre-calculated data and interpolate the weight; of course the potential is only admissible and consistent if this discretization is fine enough. The step in line 3 is the most expensive part of the algorithm, but it needs to be calculated only for nodes that are very close to the destination airport.

4.4 Computational Results

In this section, we benchmark the performance of our A* algorithm using potentials from Idealized Vertical Profiles (IVP) against Dijkstra's algorithm (D) and the Single Descent algorithm (SD). In case of Single Descent, our implementation tries to follow the description in [KCL18] as far as we could, filling in some gaps using our best judgement. To make the comparison fair, all algorithms use the same data structures, in particular, the same priority queue, such that the only difference is in the calculation of the potentials; for Dijkstra's algorithm, there are of course none. The programming language is C++, compiled with GCC 7.5.0. All computations were performed on a machine with 95 GB of RAM and an Intel(R) Xeon(R) Gold 5122 processor with 3.60GHz and 16.5 MB cache.

4.4.1 Instances

The airway network, the weather, and the aircraft performance data were provided by our industrial partner Lufthansa Systems. The airway network consists of 410387 waypoints, 878884 airway segments, and 232 flight levels. A naive construction would result in a graph with over 95 million (410387×232) nodes and over 47 billion (878884×232×231) arcs. However, a large majority of those nodes and arcs are not flyable, for example due to the waypoints and segments not available on the corresponding altitudes, or because the segment is too short for a given climb. Furthermore, the availability of certain arcs depends on the current weight, further complicating things. In our implementation, we generate the graph dynamically, therefore it is difficult to give an absolute graph size. We use propagation functions for two aircraft models, an Airbus A320 (suitable for short-haul flights) and an Airbus A340 (used for middle- to long-haul flights), derived by interpolation from corresponding tables. Unfortunately, this data, which consists of tables with millions of entries, is only an approximation of the real performance functions. It turns out that Assumption 2 is prevalent, but not always satisfied. This breaks consistency of the IVP algorithm such that it does not necessarily find the same solution as Dijkstra's. However, our computational results show that the resulting gap between Dijkstra's and the IVP A*-algorithm is mostly extremely small or non-existent, i.e., this data problem is marginal.

The OD-pairs were defined in the same way as in [KCL18]. For the long-haul test set, we chose a set of 20 major airports evenly distributed around the globe. All pairs with great-

Algorithm 1 Potential calculation

```
Input: $w, h, d$, max. descent distance function $d^{max}(\cdot)$, preprocessed step-climb- and final
    descent data.
 1: $w_0 = w$
 2: if $d < d^{max}(h)$ then
 3:     Calculate the IVP from this point by evaluating all possible step climbs followed by
            on-the-fly final descent iterations. If the distance is too short, do a simple descent.
 4:     $w \leftarrow w-$IVP consumption
 5: else
 6:     Climb to the highest level $h_1$ that is reachable and satisfies
                $$d - \text{climb distance} \leq d^{max}(h_1)$$
 7:     $w \leftarrow w-$climb consumption
 8:     $d \leftarrow d-$climb distance
 9:     $h \leftarrow h_1$
10:     if it's not possible to climb further then
11:         Read from the precalculated results what is the maximal weight on this level that
                allows a climb. Cruise until that weight is reached.
12:         $w \leftarrow w-$cruise consumption
13:         $d \leftarrow d-$cruise distance
14:         There is a set of pre-calculated step-climbs starting at the $h$ with weight $w$.
15:         Choose the maximal $h_2$ such that the step-climb to $h_2$ satisfies
                $$d - \text{step-climb distance} \leq d^{max}(h_2)$$
16:         $w \leftarrow w-$step climb consumption
17:         $d \leftarrow d-$step climb distance
18:         $h \leftarrow h_2$
19:     Cruise until $d-$cruise distance $= d^{max}(h)$
20:     $w \leftarrow w-$cruise weight
21:     $d \leftarrow d-$cruise distance
22:     In the weight discretization, find the closest weights $w_1, w_2$ s.t. $w_1 \leq w \leq w_2$
23:     Let $c_1$ be the pre-computed consumption for $w_1, h$ and $c_2$ the pre-computed con-
    sumption for $w_2, h$.
24:     $w \leftarrow w - \frac{w-w_1}{w_2-w_1}c_2 + \frac{w_2-w}{w_2-w_1}c_1$
25: return $w_0 - w$
```

circle-distances between 4000km and 11000km were considered, resulting in 202 ordered pairs. For the short-haul test set we did the same thing on the basis of a set of 19 major airports in Europe, using 500km and 4000km as distance bounds. This results in 294 ordered pairs. We calculate the short-haul flights with the A320 and the long-haul ones with the A340.

To define the take-off weight, we run Dijkstra's algorithm once on each instance, starting with the maximum possible amount of fuel. We multiply the resulting consumption by 1.2 and fix this number as the amount of fuel at take-off.

4.4.2 Methodology

As is customary in the shortest-path literature, we separate runtimes into two categories: Those in a preprocessing phase, which is instance-independent, and those in a query phase, which includes the shortest path calculation and all instance-dependent preprocessing stages. We ignore the runtime of procedures that are identical across all variants. This includes the construction of the graph, the initialization of the search algorithm, etc. More precisely: Dijkstra's algorithm (D) does not need any preprocessing. For Single Descent (SD), we consider the calculation of the minimum cruise consumption on each arc as a preprocessing operation, as dependent only on the aircraft and the weather forecast, but not on the OD-pair. The backwards search to determine the potentials is included in the query time. Idealized Vertical Profiles (IVP) require a substantial preprocessing phase for the pre-calculation of step-climbs and final-descent stages, for which we choose a weight discretization with steps of 1000kg. This preprocessing effort depends only on the aircraft, but not on the weather forecast, and not on the OD-pair. It therefore can be done once for each aircraft, which makes the associated preprocessing time irrelevant. For the sake of completeness, we nevertheless report it. As with SD, the backwards search to determine the minimum distance from each node to the destination is included in the query time. Both SD and IVP require the calculation of lower bounds on the air distance for which we use the super-optimal wind technique from [Bla+16d]. Since these computations are identical for both algorithms, we omit them. We run each calculation thrice and report the smallest time.

4.4.3 Results

Figures 4.3 and 4.4 show the query times of all three algorithms. The results are summarized in Tables 4.1 and 4.2 (short-haul and long-haul instance sets, respectively). For each statistic, we list both the arithmetic mean (*ar mean*) and the geometric mean (*geo mean*). The names used for the statistics are self-explanatory with the possible exception of *nr. labels*. This is the total count of labels that were expanded during the search.

Table 4.1: Computational results on the short-haul instances

	D		SD		IVP	
preprocessing (s)	-		0.19		11.16	
	ar mean	geo mean	ar mean	geo mean	ar mean	geo mean
query (s)	11.11	8.71	1.56	1.20	1.47	1.05
cost (kg)	4096.26	3698.07	4107.08	3709.44	4096.27	3698.07
nr. labels	472528.44	379276.36	50298.22	33548.52	41580.47	19310.30
query speedup w.r.t. D (s)	-	-	9.55	7.33	9.64	7.44
query speedup w.r.t. D ($\times$)	-	-	8.06	7.24	9.72	8.33
cost gap (kg)	-	-	10.81	0.00	0.01	0.00
cost gap (%)	-	-	0.31	0.00	0.00	0.00
labels (% of D)	-	-	10.09	8.85	7.11	5.09

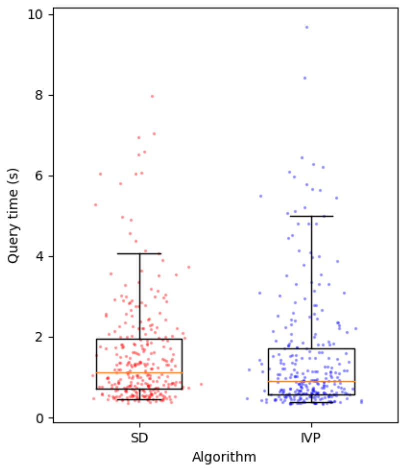


Figure 4.3: Short-haul runtimes

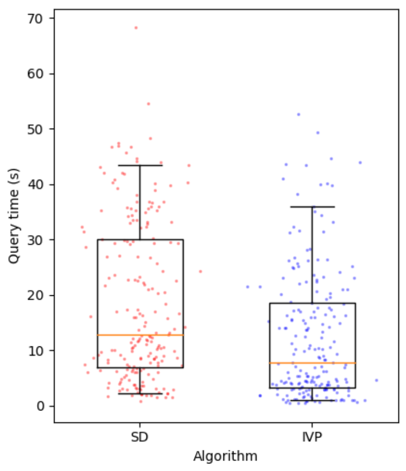


Figure 4.4: Long-haul runtimes

Table 4.2: Computational results on the long-haul instances

	D		SD		IVP	
preprocessing (s)	-		0.20		71.86	
	ar mean	geo mean	ar mean	geo mean	ar mean	geo mean
query (s)	65.78	57.03	18.52	12.87	12.33	7.30
cost (kg)	57086.24	55678.96	57100.34	55693.62	57099.54	55690.96
nr. labels	2637588.37	2340711.69	686281.66	504259.75	418759.61	243834.10
query speedup w.r.t. D (s)	-	-	47.26	41.08	53.45	46.77
query speedup w.r.t. D (×)	-	-	5.36	4.43	11.36	7.81
cost gap (kg)	-	-	14.09	0.00	13.30	0.00
cost gap (%)	-	-	0.03	0.00	0.02	0.00
labels (% of D)	-	-	24.84	21.54	14.56	10.42

The query times of both SD and IVP are far superior to Dijkstra's algorithm. Furthermore, IVP outperforms SD by roughly 5-12% on the short-haul instances and by 33-40% on the long-haul instances. As one would expect, the number of labels expanded by IVP is much smaller than that of the other two algorithms. This reduction is so significant that the expensive potential calculations are compensated. The cost of these calculations can best be seen by observing the number of labels expanded in the long-haul instances. IVP expands around 243k labels on average (geometric mean), which is less than half of those expanded by SD, while the speedup is 1.76 (geometric mean).

As can be seen in Figures 4.3 and 4.4, but also in the tables, the speedup of IVP w.r.t SD is much more pronounced in the long-haul instances. This is to be expected for various reasons: One is that in SD, both the cruise consumption estimation and the descent consumption are much nearer to the actual consumptions when flying close to the destination airport, which is the case for a big part of the search on short-haul flights. Another reason is that IVP does more expensive calculations in the area close to the destination airport — the proportion of this area to the whole search space is much larger in the short-haul case.

The preprocessing time of IVP (72s in the long-haul case) is definitely longer than that of SD but still very manageable, especially considering that it needs to be done only once per aircraft model. In practice, airlines acquire new aircraft so seldom that even a preprocessing time of several days would be acceptable.

Concerning the quality of the solutions: As expected (see Section 4.4.1), the gap[k] between the values returned by IVP and Dijkstra is not always zero, meaning that, for the data available to us, the IVP potentials are not consistent. Nevertheless, both IVP and SD yield results of a very good quality. For both variants and both test cases, the geometric mean of the gap w.r.t. Dijkstra is 0.00%, meaning that the gap is extremely small except for a few outliers. Finally, the arithmetic mean shows a small improvement of IVP over SD, especially on short-haul instances.

Another possible reason is the weight discretization used for calculating the consumption in the last section of the IVPs. However, a discretization of 1kg instead of the 1000kg used in our calculations did not yield a noticeable improvement in the solutions' quality, while slightly increasing the runtime of both queries and preprocessing. Thus, it is not included in the presented results.

4.5 Conclusion

In this paper, we investigated the Flight Planning Problem (FPP), which is a generalization of the Time-Dependent Shortest-Path Problem (TDSPP). We presented the first proof of its NP-hardness, which extends to a more general family of TDSPP variants.

We also introduced an A* algorithm based on potentials derived from Idealized Vertical Profiles (IVPs). We showed that, under reasonable theoretical assumptions on the aircraft performance functions, IVP potentials are both admissible and consistent, such that a corresponding A* algorithm finds the same solution as Dijkstra's algorithm. We show that IVP potentials can be calculated efficiently by a combination of preprocessing and on-the-fly computations.

Our computational results on real-world instances show that the effort to calculate IVP potentials pays off and results in a significant improvement of the overall query time as compared to the state-of-the-art Single Descent algorithm introduced in [KCL18]. Indeed, we obtain a speed-up of up to 40% and a smaller consistency gap.

References

[Bas+15b] Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. *Route Planning in Transportation Networks.* 2015. DOI: `10.48550/ARXIV.1504.05140`.

[Bau+20] Moritz Baum, Julian Dibbelt, Dorothea Wagner, and Tobias Zündorf. "Modeling and Engineering Constrained Shortest Path Algorithms for Battery Electric Vehicles". In: *Transportation Science* 54 (Nov. 2020), pp. 1571–1600. DOI: `10.1287/trsc.2020.0981`.

[k]We do not say *optimality gap* since Dijkstra is not guaranteed to be optimal due to the NP-hardness of the problem.

[Bla+16d] Marco Blanco, Ralf Borndörfer, Nam Dũng Hoàng, Anton Kaier, Adam Schienle, Thomas Schlechte, and Swen Schlobach. "Solving Time Dependent Shortest Path Problems on Airway Networks Using Super-Optimal Wind". In: *16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2016)*. Vol. 54. This work is licensed under the Creative Commons Attribution 3.0 Unported License. To view a copy of this license, visit `http://creativecommons.org/licenses/by/3.0/`. 2016. DOI: `10.4230/OASIcs.ATMOS.2016.12`.

[Bla+16e] Marco Blanco, Ralf Borndörfer, Nam Dũng Hoàng, Anton Kaier, Thomas Schlechte, and Swen Schlobach. *The Shortest Path Problem with Crossing Costs*. eng. Tech. rep. Berlin, 2016.

[Bla+17b] Marco Blanco, Ralf Borndörfer, Nam Dung Hoàng, Anton Kaier, Pedro M. Casas, Thomas Schlechte, and Swen Schlobach. "Cost Projection Methods for the Shortest Path Problem with Crossing Costs". In: *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017)*. Ed. by Gianlorenzo D'Angelo and Twan Dollevoet. Vol. 59. OpenAccess Series in Informatics (OASIcs). This work is licensed under the Creative Commons Attribution 3.0 Unported License. To view a copy of this license, visit `http://creativecommons.org/licenses/by/3.0/`. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, 15:1–15:14. ISBN: 978-3-95977-042-2. DOI: `10.4230/OASIcs.ATMOS.2017.15`. URL: `http://drops.dagstuhl.de/opus/volltexte/2017/7893`.

[DN12] Daniel Delling and Giacomo Nannicini. "Core Routing on Dynamic Time-Dependent Road Networks". In: *INFORMS Journal on Computing* 24.2 (May 2012), pp. 187–201. DOI: `10.1287/ijoc.1110.0448`.

[EFS11] Jochen Eisner, Stefan Funke, and Sabine Storandt. "Optimal Route Planning for Electric Vehicles in Large Networks". In: *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*. AAAI'11. San Francisco, California: AAAI Press, 2011, pp. 1108–1113.

[He+20] Edward He, Natashia Boland, George Nemhauser, and Martin Savelsbergh. "Time-Dependent Shortest Path Problems with Penalties and Limits on Waiting". In: *INFORMS Journal on Computing* (2020).

[HM98] Patrick Hagelauer and Felix Antonio Claudio Mora-Camino. "A soft dynamic programming approach for on-line aircraft 4D-trajectory optimization". In: *European Journal of Operational Research* 107.1 (May 1998), pp. 87–95. DOI: `10.1016/S0377-2217(97)00221-X`. URL: `https://hal-enac.archives-ouvertes.fr/hal-01021633`.

[JCL17b] Casper Kehlet Jensen, Marco Chiarandini, and Kim S. Larsen. "Flight Planning in Free Route Airspaces". In: *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017)*. Ed. by Gianlorenzo D'Angelo and Twan Dollevoet. Vol. 59. OpenAccess Series in Informatics (OASIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, pp. 1–14. ISBN: 978-3-95977-042-2. DOI: `10.4230/OASIcs.ATMOS.2017.14`.

[Jon74] H.M. de Jong. *Optimal Track Selection and 3-dimensional flight planning*. Tech. rep. 1974.

[Kar+12a] Stefan E. Karisch, Stephen S. Altus, Goran Stojković, and Mirela Stojković. "Operations". English. In: *Quantitative Problem Solving Methods in the Airline Industry.* Ed. by Cynthia Barnhart and Barry Smith. Vol. 169. International Series in Operations Research & Management Science. Springer US, 2012, pp. 283–383. ISBN: 978-1-4614-1607-4.

[KCL16] Anders Nicolai Knudsen, Marco Chiarandini, and Kim S. Larsen. "Vertical Optimization of Resource Dependent Flight Paths". In: *ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016).* 2016, pp. 639–645. DOI: 10.3233/978-1-61499-672-9-639.

[KCL17] Anders Nicolai Knudsen, Marco Chiarandini, and Kim S. Larsen. "Constraint Handling in Flight Planning". In: *Principles and Practice of Constraint Programming - 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings.* 2017, pp. 354–369. DOI: 10.1007/978-3-319-66158-2_23. URL: https://doi.org/10.1007/978-3-319-66158-2_23.

[KCL18] Anders Nicolai Knudsen, Marco Chiarandini, and Kim S. Larsen. "Heuristic Variants of A* Search for 3D Flight Planning". In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 15th International Conference, CPAIOR 2018, Delft, The Netherlands, June 26-29, 2018, Proceedings.* 2018, pp. 361–376. DOI: 10.1007/978-3-319-93031-2_26. URL: https://doi.org/10.1007/978-3-319-93031-2_26.

[Mar+21] Pedro Maristany de las Casas, Luitgard Kraus, Antonio Sedeño-Noda, and Ralf Borndörfer. *Targeted Multiobjective Dijkstra Algorithm.* 2021. DOI: 10.48550/ARXIV.2110.10978. URL: https://arxiv.org/abs/2110.10978.

[MSB21] P. Maristany de las Casas, A. Sedeño-Noda, and R. Borndörfer. "An improved Multiobjective Shortest Path algorithm". In: *Computers & Operations Research* (June 2021), p. 105424. DOI: 10.1016/j.cor.2021.105424.

[NSG14] Hok K. Ng, Banavar Sridhar, and Shon Grabbe. "Optimizing Aircraft Trajectories with Multiple Cruise Altitudes in the Presence of Winds". In: *Journal of Aerospace Information Systems* 11.1 (2014), pp. 35–47.

[NU02] Matti Nykänen and Esko Ukkonen. "The Exact Path Length Problem". In: *Journal of Algorithms* 42.1 (2002), pp. 41–53. ISSN: 0196-6774. DOI: https://doi.org/10.1006/jagm.2001.1201. URL: https://www.sciencedirect.com/science/article/pii/S0196677401912015.

[OR89] Ariel Orda and Raphael Rom. *Traveling without Waiting in Time-Dependent Networks is NP-Hard.* Tech. rep. Department Electrical Engineering, Technion-Israel Institute of Technology, 1989.

[SMB19] Adam Schienle, Pedro Maristany, and Marco Blanco. "A Priori Search Space Pruning in the Flight Planning Problem". In: *19th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2019).* Ed. by Valentina Cacchiani and Alberto Marchetti-Spaccamela. Vol. 75. OpenAccess Series in Informatics (OASIcs). This work is licensed under the Creative Commons Attribution 3.0 Unported License. To view a copy of this license, visit http://creativecommons.org/licenses/by/3.0/. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, 8:1–8:14. ISBN: 978-3-95977-128-3. DOI: 10.4230/OASIcs.ATMOS.2019.8.

[SZ19] Ben Strasser and Tim Zeitz. *A Fast and Tight Heuristic for A* in Road Networks.* 2019. DOI: `10.48550/ARXIV.1910.12526`.

[Zei23] Tim Zeitz. “NP-hardness of shortest path problems in networks with non-FIFO time-dependent travel times”. In: *Information Processing Letters* 179 (2023), p. 106287. ISSN: 0020-0190. DOI: `https://doi.org/10.1016/j.ipl.2022.106287`. URL: `https://www.sciencedirect.com/science/article/pii/S0020019022000448`.

Introduction to Chapter 5

This chapter was published in [Bla+17b].

In the Flight Planning literature, overflight charges are often overlooked or overly simplified. The only serious analysis we are aware of is [Bla+16e], which is also our work but is only published as a technical report. This lack of attention is surprising, since flight routes in practice are significantly affected by overflight costs, despite these usually being smaller than fuel and time costs. The underlying mathematical problems arising from the various cost models are also fascinating, see [Bla+16e] for more details.

A cost model that stands out for being both mathematically challenging and very relevant in practice is the one based on Great Circle Distance (GCD) and linear costs. Challenging because costs depend on the full path and are not defined on individual arcs. This makes finding guaranteed optimal solutions in a time that is acceptable for real-world applications seems to be impossible. Relevant because this cost model is used in all of Europe and plenty of other high-traffic countries, such as the USA and Canada. This is the cost model that we consider in this chapter.

This chapter presents a preprocessing algorithm that generates simple arc-based costs that, on a route by route basis, aim to approximate the real costs. Although, to the best of our knowledge, this algorithm is not able to provide an a priori approximation guarantee, our tests indicate an error of less than 1%. For making these comparisons, we present the *Two-Layer Dijkstra*, an exact algorithm to solve this problem. Its detailed description is included in Appendix 5.8. Note that this appendix is not part of the original publication.

Although the model considered in this chapter seems restrictive (no consideration of weather or fuel, only horizontal optimization), overflight charges in practice are dependent only on the horizontal route. These techniques are completely independent of the other components of the problem and can be trivially combined with any of the algorithms presented in the other chapters.

References

[Bla+16e] Marco Blanco, Ralf Borndörfer, Nam Dũng Hoàng, Anton Kaier, Thomas Schlechte, and Swen Schlobach. *The Shortest Path Problem with Crossing Costs.* eng. Tech. rep. Berlin, 2016.

[Bla+17b] Marco Blanco, Ralf Borndörfer, Nam Dung Hoàng, Anton Kaier, Pedro M. Casas, Thomas Schlechte, and Swen Schlobach. “Cost Projection Methods for the Shortest Path Problem with Crossing Costs”. In: *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017).* Ed. by Gianlorenzo D’Angelo and Twan Dollevoet. Vol. 59. OpenAccess Series in Informatics (OASIcs). This work is licensed under the Creative Commons Attribution 3.0 Unported License. To view a copy of this license, visit `http://creativecommons.org/licenses/by/3.0/`. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, 15:1–15:14. ISBN: 978-3-95977-042-2. DOI: `10.4230/OASIcs.ATMOS.2017.15`. URL: `http://drops.dagstuhl.de/opus/volltexte/2017/7893`.

Chapter 5

Cost Projection Methods for the Shortest Path Problem with Crossing Costs

Marco Blanco, Ralf Borndörfer, Nam-Dũng Hoang, Anton Kaier, Pedro Maristany de las Casas, Thomas Schlechte, Swen Schlobach

Abstract

Real world routing problems, e.g., in the airline industry or in public and rail transit, can feature complex non-linear cost functions. An important case are costs for crossing regions, such as countries or fare zones. We introduce the *shortest path problem with crossing costs* (SPPCC) to address such situations; it generalizes the classical shortest path problem and variants such as the resource constrained shortest path problem and the minimum label path problem.

Motivated by an application in flight trajectory optimization with *overflight costs*, we focus on the case in which the crossing costs of a region depend only on the nodes used to enter or exit it. We propose an exact *Two-Layer-Dijkstra Algorithm* as well as a novel *cost-projection* linearization technique that approximates crossing costs by shadow costs on individual arcs, thus reducing the SPPCC to a standard shortest path problem. We evaluate all algorithms' performance on real-world flight trajectory optimization instances, obtaining very good à posteriori error bounds.

5.1 Introduction

In this work, we introduce the shortest path problem with crossing costs (SPPCC), which seeks to find a minimum-cost path between two given nodes in a directed graph. As opposed to the classical shortest path problem (SPP), where the objective function is the sum of weights corresponding to the arcs of the path, we also consider *crossing costs*. These are associated with regions in the digraph, which are defined as sets of arcs. Every time a path intersects a region, the corresponding crossing costs are determined using the region's cost function, which depends on the arcs in the intersection. Crossing costs come up as overflight costs in aircraft routing, where the regions correspond to countries [Kar+12a], in passenger routing in public and rail transport, where fare zones and air distance dependent tariffs are common [HS04], and in intermodal routing, when modes are changed [Bar+08].

The SPPCC constitutes a generalization of the classical SPP, see [Som14] for an excellent survey, and of several NP-hard variants such as the resource constrained shortest path problem and the minimum label path problem [GJ79]. While the first can be approximated in polynomial time [LR01], the second cannot within a polylogarithmic factor unless P=NP [HMS07]. Exact algorithms for the resource constrained shortest path problem have been suggested by [ID05], and for a simplified version of the minimum label path problem, the problem of finding a path with the smallest number of different colors in an arc-colored graph, by [Bro+05].

We focus on a variant of the SPPCC that arises from assuming linear crossing costs that depend only on the first and last nodes used in each region. To the best of our knowledge, this problem has not yet been studied in the optimization community. The paper makes the following contributions:

1. The *Two-Layer-Dijkstra Algorithm*, which solves the problem to optimality in polynomial time.

2. Two *cost-projection* methods (one combinatorial, one LP-based) that transform crossing costs into regular arc weights, these are used to transform the problem into a related standard shortest path problem.

3. A computational evaluation of all algorithms presented.

We will show that cost projection allows to deal with complex crossing costs in a way that is highly accurate, computationally efficient, and allows to deal with changes of arc weights according to, e.g., weather.

Our motivation for studying the SPPCC stems from an application in flight trajectory optimization, as described in [Kar+12a]. This problem can be described as follows. The input is an airway network, origin and destination airports, an aircraft of a given weight, a weather forecast, and countries' overflight fees functions, among other factors. The output is a flight route. The objective is to minimize the trajectory-dependent costs (as opposed to constant costs such as airport fees), which are usually the sum of fuel costs, overflight costs, and time costs. Fuel costs and time costs are in general directly proportional to the length of the trajectory, and can thus, after some simplifications, be projected down onto the airway segments (i.e., the arcs of a graph representation), essentially reducing the problem to a (time-dependent) shortest-path problem, which can be solved to optimality by using a variant of Dijkstra's algorithm. Overflight fees, however, are determined by very diverse cost models, many of which make Dijkstra's algorithm deliver suboptimal solutions.

Overflight costs, also known as *ATC charges*, are the means by which Air Traffic Control authorities finance themselves. For each *cost airspace* (in general corresponding to a country) used during a flight, airlines are required to pay a fee, determined by factors such as the type of aircraft and the way in which the airspace is overflown. To the best of our knowledge, all currently used models define the overflight cost on an airspace as a function that takes three parameters: The distance defined by the flight trajectory in the airspace, the aircraft's maximum take-off weight, and the origin-destination pair. Since the last two are independent of the trajectory, we will assume from here on that the overflight cost function is solely dependent on the distance.

There exist two widely used definitions of the distance determined by the intersection of a flight trajectory and an airspace. The first and most natural variant is to consider the *flown distance* (FD), which is the total ground distance traversed by the aircraft in the airspace. The second variant uses the *great-circle-distance* (GCD), defined in this context as the length of the great circle connecting the coordinates where the aircraft enters and exits the airspace. If a trajectory intersects an airspace multiple times, the distance considered is the sum of the distances defined by each intersection.

The cost function itself is always non-decreasing and usually linear, constant, or piecewise-constant. This results in six different combinations, which encompass nearly all cost models currently in use, and which lead to problems with varying degrees of difficulty. The only outliers, as of April 2016, are India, Argentina, Philippines, Democratic Republic of Congo, and Kenya; where the function used is piecewise-linear or even non-linear. In the remainder of this paper, we will ignore these cases. In Table 5.1 we list some representative airspaces which use the aforementioned models. Notice that when the cost function is constant for an airspace, it is irrelevant whether the distance type used is GCD or FD. Official documentations of overflight cost models can be found for example at [Fed16], [ASE16], or [EUR16a].

Table 5.1: Examples of airspaces and different cost models, as of April 2016.

Function \ **Distance**	**GCD**	**FD**
Linear	European countries, USA, Thailand	Brazil, Ghana, Saudi Arabia, Iran
Constant	Egypt, Sudan, Myanmar, Japan	
Piecewise-constant	Ethiopia	ASECNA[a], Angola, Vietnam
Other	India, D.R. Congo	Argentina, Kenya

The problem of flight trajectory optimization under consideration of overflight costs is essential for minimizing airlines' operational expenses. In a case reported in [Car07], an airline could save 884 USD in overflight charges on a single flight from San Francisco to Frankfurt by deviating from the standard, most direct route; thus avoiding Canada's expensive charges. However, despite their obvious importance, the literature on overflight costs is scarce. The problem is presented in [Kar+12a], but no solution approaches are discussed. The authors in [Bon+13b] consider the linear/GCD model, but their optimal control algorithm approximates overflight costs by using FD instead of GCD. Similarly, in [Sri+15], several variants of overflight charges are introduced (including linear/GCD and flat-rate), but the optimization algorithm deals only with linear/FD costs. [SPS03] introduces the linear/GCD model as well as the piecewise-constant/FD model, but their heuristic solution method allows no *à posteriori* quality assessment.

In this paper, we study the flight trajectory optimization problem with overflight costs through the more abstract *shortest path problem with crossing costs* (SPPCC). We present the first formal framework for classification of existing overflight cost models. Furthermore, we introduce efficient algorithms to handle a problem variant that is one of the most relevant in practice. The paper is structured as follows. In Section 5.2.1, we formally introduce the SPPCC, which models the problem of flight trajectory optimization with overflight costs. In Sections 5.3 and 5.4, we focus on one of the most important variants of the SPPCC. We present both the Two Layer Dijkstra Algorithm, a polynomial, exact algorithm; as well as two novel cost-projection techniques that reduce the problem (heuristically) to a standard shortest path

[a]ASECNA is a cost airspace encompassing 18 African countries, with an area of approximately 1.5 times that of Europe.

problem. In Section 5.5 we present computational results on real-world data.

5.2 Problem Description

This section gives a formal definition of the shortest path problem with crossing costs. For consistency with the literature, we will first introduce a general, NP-hard version and then proceed for the remainder of the paper with a polynomially solvable version, which is relevant for the applications in aircraft (and intermodal) routing that we have in mind.

5.2.1 The General SPPCC

Let $D = (V, A)$ be a directed graph, with source and target nodes s and t. Let $d : V \times V \to [0, \infty)$ be a metric function that we call *distance*. For the special case where $(u, v) = a \in A$, we use the notation $d_a := d(u, v)$. We also have a constant factor $\varphi > 0$, which represents the unit weight on arcs. For each $a \in A$, we refer to $\varphi_a := \varphi \cdot d_a$ as the *weight* of arc a.

Let $\mathcal{R} = \{R_1, \ldots, R_k\} \subseteq 2^A$ be a family of sets of arcs, not necessarily disjoint, that we shall call *regions*, such that $\bigcup_{R \in \mathcal{R}} R = A$. Furthermore, let $\mathcal{R}^G, \mathcal{R}^F \subseteq \mathcal{R}$ be a partition of $\mathcal{R}$. $\mathcal{R}^F$ represents the regions where every arc belonging to a solution path is important for computing the costs. For regions in $\mathcal{R}^G$, only the nodes used to enter and exit the region are relevant. For every $R \in \mathcal{R}$, let $f_R : [0, \infty) \to [0, \infty)$ be a non-decreasing function. For a path p in D and a region $R \in \mathcal{R}$, let $\mathcal{P}^p_R$ be the set of all maximal subpaths of p contained in R. The nodes $t(p)$ and $h(p)$ denote the first and last nodes in a path p, respectively. Similarly, for an arc $a = (u, v)$, we use the notation $t(a) = u, h(a) = v$.

Finally, we can define the *crossing cost* corresponding to $R \in \mathcal{R}$ and an (s, t)-path p:

$$\gamma_R(p) = \begin{cases} f_R\left(\sum_{q \in \mathcal{P}^p_R} d\left(t(q), h(q)\right)\right) & \text{if } R \in \mathcal{R}^G \text{ and } R \cap p \neq \emptyset \\ f_R\left(\sum_{a \in R \cap p} d_a\right) & \text{if } R \in \mathcal{R}^F \text{ and } R \cap p \neq \emptyset \\ 0 & \text{if } R \cap p = \emptyset \end{cases}$$

Definition 5.1. The *shortest path problem with crossing costs* (SPPCC) is defined as follows:
Input: Directed graph $D = (V, A)$, arc weights $\varphi : A \geq \mathbb{R}_+$, regions $\mathcal{R} \subseteq 2^A$, crossing costs γ, nodes $s, t \in V$.
Objective: Compute an (s, t)-path p in D that minimizes the cost function

$$c(p) = \sum_{a \in p} \varphi_a + \sum_{R \in \mathcal{R}} \gamma_R(p). \tag{5.1}$$

Example 5.1. We illustrate the definition of the SPPCC with the example in Figure 5.1. There, we have regions $\mathcal{R} = \{R^1, R^2, R^3, R^4, R^5, R^6\}$, with the corresponding crossing cost functions given in Table 5.2. In this example, we define the arc weights as $\varphi_a := 2d_a$ for each $a \in A$. We also suppose $R^2, R^4 \in \mathcal{R}^G$; $R^1, R^5 \in \mathcal{R}^F$. Since f_{R^3} and f_{R^6} are constant, it is irrelevant whether they belong to $\mathcal{R}^G$ or $\mathcal{R}^F$, but for the sake of completeness let us say $R^3, R^6 \in \mathcal{R}^F$. Given that R^2 and R^4 belong to $\mathcal{R}^G$, we use the distances between the first and last nodes of the intersecting subpaths to evaluate the crossing cost functions. The total cost of the example

Table 5.2: Crossing cost functions for Example 5.1.

i	1	2	3	4	5	6
$f_{R^i}(x)$	$3x$	$2x$	10	$\begin{cases} 8 & \text{if } x \leq 6 \\ 20 & \text{if } x > 6 \end{cases}$	$\begin{cases} 5 & \text{if } x \leq 10 \\ 10 & \text{if } x > 10 \end{cases}$	15

path is thus:

$$\begin{aligned} c(p) =& \sum_{a \in p} \varphi_a + \gamma_{R^1}(p) + \gamma_{R^2}(p) + \gamma_{R^3}(p) + \gamma_{R^4}(p) + \gamma_{R^5}(p) + \gamma_{R^6}(p) \\ =& 88 + f_{R^1}(9) + f_{R^2}(4) + f_{R^3}(7) + f_{R^4}(10) + f_{R^5}(10) + 0 = 158. \end{aligned}$$

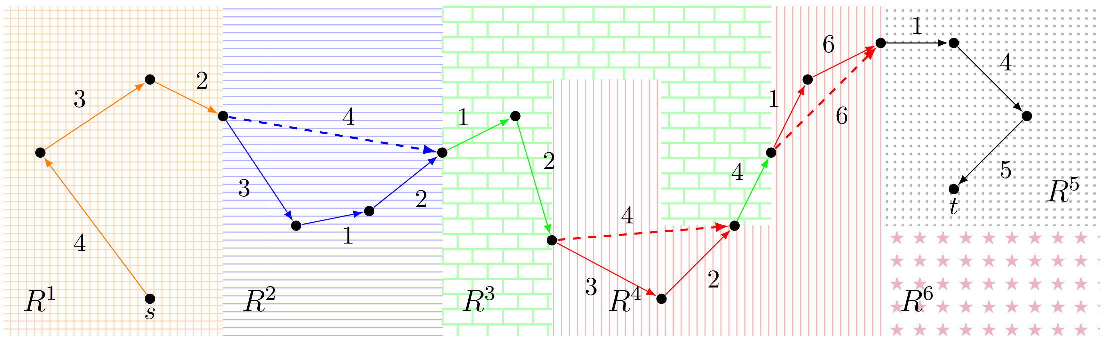


Figure 5.1: Example of an SPPCC instance with six regions. Arcs are labeled with the distances d_a between their endpoints, and the (s,t)-path p is represented by the continuous arrows. Dashed arrows represent distances between nodes that are not connected by an arc in the path.

The problem of flight trajectory optimization with overflight costs, as described in Section 5.1, can be modeled using the SPPCC as follows: We use V to represent waypoints and A to represent airway segments. The nodes s and t represent the departure and destination airports. An airspace, which is defined as a geographical region, can be modeled by a set $R \in \mathcal{R}$. For each $a \in A$, the cost φ_a can be seen as the fuel cost corresponding to traversing airway segment a, multiplied by a wind distortion that will be explained in the next subsection. Furthermore, the great-circle-distance between two points u and v is represented by the function $d(u,v)$. $\mathcal{R}^G$ represents the set of airspaces that use the GCD-model for measuring distance, $\mathcal{R}^F$ those that use FD.

The five important airspace cost models outlined in Section 5.1 induce several different variants of the SPPCC. An extensive analysis of their complexity can be found in [Bla+16c]. In this paper, we will focus on a variant which is of particular interest in our application.

5.2.2 The SPPCC With Wind Influence

In the remainder of the paper, we will focus on a case that best matches our application and turns out to be polynomial. We assume that $\mathcal{R}^G = \mathcal{R}$, $\mathcal{R}^F = \emptyset$, and that f_R is linear for all $R \in \mathcal{R}$. This allows to model the overflight fees used, for example, in Canada, USA, and most European states. We also assume that the regions in R are pairwise disjoint. This is natural, since usually there is a one-to-one correspondence between cost airspaces and countries, and the latter do not overlap. What about the weather? Of course, overflight fees do not depend

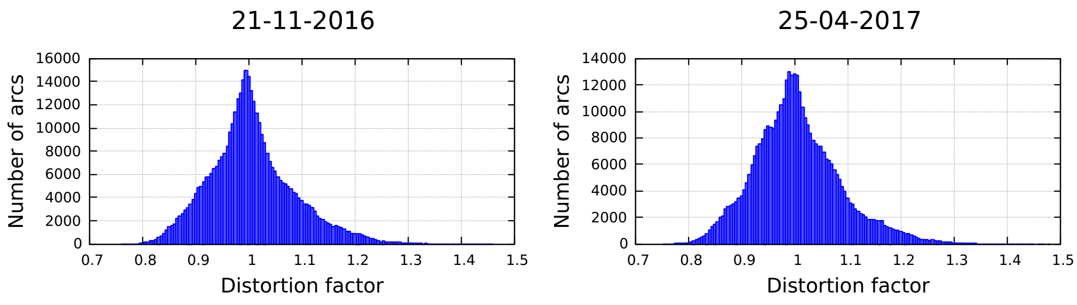


Figure 5.2: Typical distributions of the weight distortion factors ω_a over all arcs, for two different weather prognoses (November 2016 on the left, April 2017 on the right).

on the weather, but the time needed to fly between two nodes (and thus the corresponding fuel consumption) varies greatly as the wind speed and direction change with time, see [Bla+16d] for a more detailed discussion of this phenomenon. Luckily, these tow cost factors can be separated in such a way that we can deal with crossing costs in *in a preprocessing step independently of the weather*, and with the wind *at query time* by multiplying each arc weight φ_a with an appropriate *distortion factor* ω_a. Figure 5.2 tries to provide some intuition on the distribution of these factors in our application. Motivated by this data, we will assume $\omega_a \in [0.7, 1.4]$ $\forall a \in A$ throughout the paper.[a] We refer to the resulting variant of the SPPCC as SPPCC-W:

Definition 5.2. The *shortest path problem with crossing costs and wind influence* (SPPCC-W) is defined as follows:
Input at preprocessing time: Directed graph $D = (V, A)$, arc weights $\varphi : A \geq \mathbb{R}_+$, regions $\mathcal{R} \subseteq 2^A$, crossing costs γ.
Input at query time: Nodes $s, t \in V$, distortion factors $\omega : A \to [0.7, 1.4]$.
Objective: Compute an (s, t)-path p in D that minimizes the cost function

$$c(p) = \sum_{a \in p} \omega_a \varphi_a + \sum_{R \in \mathcal{R}} \gamma_R(p). \tag{5.2}$$

5.3 The Two-Layer-Dijkstra Algorithm

We introduce in this section the *Two-Layer-Dijkstra Algorithm* (2LDA), which solves the SPPCC-W to optimality in polynomial time. Recall that $\mathcal{R}^G = \mathcal{R}$, that is, for each region, crossing costs are given by a linear function evaluated at the distance between the first and the last nodes used in that region (or the corresponding sum of distances, in case a region is intersected multiple times). The main difficulty is that two different paths crossing a region and using the same nodes to enter and leave it incur the same crossing costs. It can be overcome by considering a *coarse* graph on a subset of the nodes of the original (*fine*) graph, whose arcs correspond to shortest paths traversing a region in the fine graph. We run a Dijkstra algorithm on the coarse graph, determining arc costs on-the-fly by repeatedly using Dijkstra's algorithm on the fine graph. Finally, from the optimal path in the coarse graph, the minimum-cost path in the fine graph is reconstructed. We recall that, in Section 5.2.2, we assumed that the arc weights are multiplied with a distortion factor after preprocessing. Since the 2LDA does not have a preprocessing phase, we will ignore distortion in this section, without loss of generality.

[a]We do not use these bounds explicitly, but the knowledge that ω_a is always "close to 1" is an important motivation for our algorithms in Section 5.4.

Let p be a path in D, whose intersections with a region $R \in \mathcal{R}$ are the subpaths $p_1^R, \dots, p_r^R$. By the assumption of linearity of all crossing cost functions, there exists $\alpha_R > 0$ such that the crossing costs of p in R can be written as $\gamma_R(p) = \alpha_R \cdot \big(d(t(p_1^R), h(p_1^R)) + \cdots + d(t(p_r^R), h(p_r^R))\big)$. Recall that $t(\cdot)$ and $h(\cdot)$ denote a path's first and last nodes, or an arc's tail- and head nodes.

Some more definitions are necessary before introducing the algorithm. For $R \in \mathcal{R}$, define the sets of nodes that can be used to enter or exit R as follows:

$$V^{\text{entry}}(R) := \begin{cases} \{v \in V | \delta^-(v) \backslash R \neq \emptyset, \delta^+(v) \cap R \neq \emptyset\} \cup \{s\} & \text{if } \delta^+(s) \cap R \neq \emptyset \\ \{v \in V | \delta^-(v) \backslash R \neq \emptyset, \delta^+(v) \cap R \neq \emptyset\} & \text{if } \delta^+(s) \cap R = \emptyset \end{cases}$$

$$V^{\text{exit}}(R) := \begin{cases} \{v \in V | \delta^-(v) \cap R \neq \emptyset, \delta^+(v) \backslash R \neq \emptyset\} \cup \{t\} & \text{if } \delta^-(t) \cap R \neq \emptyset \\ \{v \in V | \delta^-(v) \cap R \neq \emptyset, \delta^+(v) \backslash R \neq \emptyset\} & \text{if } \delta^-(t) \cap R = \emptyset \end{cases},$$

where $\delta^-(v)$ is the set of arcs incident to v and $\delta^+(v)$ is the set of arcs going out from v. $V^{\text{entry}}(R)$ and $V^{\text{exit}}(R)$ will be referred to as the sets of *entry* and *exit* nodes of R, respectively. We assume that for $R \in \mathcal{R}$ we have $V^{\text{entry}}(R) \cap V^{\text{exit}}(R) = \emptyset$. If this is not the case, we can achieve it through a simple transformation that duplicates nodes and assigns incident arcs in an appropriate way, which does not affect the hardness of the problem.

Consider a directed graph $\bar{D} = (\bar{V}, \bar{A})$, where

$$\bar{V} := \bigcup_{R \in \mathcal{R}} (V^{\text{entry}}(R) \cup V^{\text{exit}}(R)) \text{ and } \bar{A} := \bigcup_{R \in \mathcal{R}} (V^{\text{entry}}(R) \times V^{\text{exit}}(R)).$$

Given that a node cannot simultaneously be an entry- and an exit node of the same region, and since we assume that $\mathcal{R}$ partitions A, there exists a unique region $R_{\bar{a}}$ for each $\bar{a} \in \bar{A}$ such that the tail node of $\bar{a}$ is an entry point of $R_{\bar{a}}$ and its head node an exit point. Let $D|_{R_{\bar{a}}}$ denote the subgraph of D induced by $R_{\bar{a}}$. We define the function COMPUTECOST($\bar{a}$) as follows, for every $\bar{a} \in \bar{A}$: First we compute the shortest $(t(\bar{a}), h(\bar{a}))$-path in $D|_{R_{\bar{a}}}$ using arc weights φ. Let $z_{\bar{a}}$ be the corresponding optimal value, or $+\infty$ if no such path could be found. COMPUTECOST($\bar{a}$) then returns $z_{\bar{a}} + f_{R_{\bar{a}}}(d(u,v))$.

Given these definitions, the *Two-Layer-Dijkstra Algorithm* is very simple. It runs in two phases, and works as follows:

Phase 1 is identical to the classical Dijkstra algorithm searching a shortest (s,t)-path on $\bar{D}$ except for one detail. Whenever an arc $\bar{a}$ is examined by the algorithm, we do not know its costs a priori. Instead, we compute them on-the-fly by calling COMPUTECOST($\bar{a}$). In Phase 2, each arc $\bar{a}$ in the optimal solution is expanded to a subpath in D by recomputing the shortest $(t(\bar{a}), h(\bar{a}))$-path in $D|_{R_{\bar{a}}}$ using arc weights φ. Finally, all such subpaths are concatenated to return an (s,t)-path in D. See Appendix 5.8 for a formal description.

Theorem 5.1. *The Two-Layer-Dijkstra Algorithm returns the optimal solution to SPPCC-W.*

Proof. Analogously to the classical Dijkstra algorithm, a simple inductive process on the number of visited nodes in $\bar{D}$ shows that the Two-Layer-Dijkstra Algorithm finds the optimal solution from s to every $\bar{v} \in \bar{V}$; and in particular to t. □

Since the algorithm does $O(|V|^2)$ iterations of the Dijkstra algorithm, it has a total running time of $O(n^2 m + n^3 \log(n))$, where $n = |V|$ and $m = |A|$. Thus, we conclude:

Corollary 5.1. SPPCC-W can be solved in polynomial time.

5.4 The Cost Projection Method

While polynomial, the 2LDA is too slow for practical purposes. This motivates the development of fast heuristics. For this purpose, we introduce what we call the *cost projection problem.* It is based on the idea of replacing crossing costs in a preprocessing phase with shadow arc weights x_a and adding these to the original arc weights φ_a, thus resulting in a related SPP instance. Intuitively, the goal is to define the arc shadow weights in such a fashion that the path hierarchy (with respect to total costs) is preserved after the cost transformation. Formally:

Definition 5.3. The *cost projection problem (CPP)* is the following:
Input: SPPCC-W instance (see Definition 5.2).
Objective: In a preprocessing phase (i.e., without knowledge of s, t and ω), construct *projected costs* $x : A \to \mathbb{R}_+$ so that, for any two paths p, q in D, it holds:

$$\sum_{a\in p} \omega_a \varphi_a + \sum_{R\in\mathcal{R}} \gamma_R(p) \leq \sum_{a\in q} \omega_a \varphi_a + \sum_{R\in\mathcal{R}} \gamma_R(q) \Leftrightarrow \sum_{a\in p}(\omega_a\varphi_a + x_a) \leq \sum_{a\in q}(\omega_a\varphi_a + x_a).$$

It is clear that, if we could successfully solve the CPP, then the SPPCC-W could be reduced to a simple SPP, and then solved very efficiently. Unfortunately, due to the enormous number of potential paths, the non-linear nature of the crossing costs, and the non-deterministic properties of the distortion factors ω, the CPP does not have a feasible solution in general. Some progress can be made by observing that, for our purposes, it would suffice if the optimal path in the original SPPCC-W instance were mapped to the optimal path in the transformed SPP instance. However, this is an equally unrealistic goal. We will therefore try to construct projected costs in such a way that the path found by solving the transformed SPP instance to optimality has a small optimality gap w.r.t. the original SPPCC-W instance. We suggest to do this by focusing on the subpaths in each region that are likely to be contained in the optimal SPPCC path.

Definition 5.4. For each region, we define a set of *good paths*: Given $R \in \mathcal{R}, r \geq 1, u \in V^{\text{entry}}(R)$ and $v \in V^{\text{exit}}(R)$, we say that a (u, v)-path p is *r-good* if

$$\sum_{a\in p} d_a \leq r \cdot d(u, v). \tag{5.3}$$

We denote the set of r-good paths in R by $\mathcal{G}_r^R$. For any subset $\mathcal{G} \subseteq \mathcal{G}_r^R$, the subset of arcs in R used by at least one path in $\mathcal{G}$ is denoted by $A(\mathcal{G})$.

Intuitively, since arc weights φ are defined as a constant multiple of arc lengths, a path is good if it is not much longer than the theoretically shortest possible path. Recalling the assumption we made in Definition 5.2 on the possible ranges for the distortion factors ω_a, it is clear that the shortest paths within an airspace at query time are likely to be good paths, after a suitable choice of the parameter r (see Section 5.5). Finally, note that the size of $\mathcal{G}_r^R$ is exponential in the number of arcs.

We propose two approaches for solving the CPP.

5.4.1 Averaging Unit Costs: A Combinatorial Cost Projection Method

The first method we discuss is purely combinatorial and very simple, we call it *Averaging Unit Costs* (`AUC`). The idea behind the approach has the following motivation:

Suppose all paths in $\mathcal{G}_r^R$ in a region R are pairwise disjoint. A path $p \in \mathcal{G}_r^R$ has length $d(p) = \sum_{a\in p} d_a$ and costs $\gamma_R(p)$. A natural way to define projected costs on an arc $a \in p$ is to do so proportionally to its length:

$$x_a^{R,p} := d_a \cdot \frac{\gamma_R(p)}{d(p)}. \tag{5.4}$$

As a consequence of regions being disjoint, an arc $a \in A$ belongs to exactly one region $R \in \mathcal{R}$. Hence, to ease notation, we omit the index R in $x_a^{R,p}$. Since, in addition, all paths in $\mathcal{G}_r^R$ are disjoint, projected costs (5.4) for each arc in a good path are unique. Particularly, for each path $p \in \mathcal{G}_r^R$, there holds

$$x(p) = \sum_{a \in p} x_a^p = \gamma_R(p). \tag{5.5}$$

If (5.5) were to hold for all paths in all regions, then the CPP would be solved. However, this assumption is not realistic. In practice, an arc $a \in \mathcal{C}^R$ is usually contained in more than one good path. Furthermore, as noticed above, the size of $\mathcal{G}_r^R$ is exponential, making the iteration over all good paths impractical. For that reason, we restrict ourselves to a polynomially-sized subset $\bar{\mathcal{G}}_r^R \subset \mathcal{G}_r^R$ (a possible way of constructing $\bar{\mathcal{G}}_r^R$ is presented in Section 5.5), and define the final projected costs x_a of an arc a as the average over the projected costs (5.4) of a derived from all paths $p \in \bar{\mathcal{G}}_r^R$ covering it. Formally:

Let $a \in A(\bar{\mathcal{G}}_r^R)$, and let $\mathcal{G}_a^R \subseteq \bar{\mathcal{G}}_r^R$ be the subset of good paths in $\bar{\mathcal{G}}_r^R$ containing a. Note that $\mathcal{G}_a^R \neq \emptyset$, since $a \in A(\bar{\mathcal{G}}_r^R)$. Then, the final projected cost x_a of a is defined as

$$x_a := \frac{\sum_{p \in \mathcal{G}_a^R} x_a^p}{|\mathcal{G}_a^R|}, \tag{5.6}$$

Note that at this point, projected costs x_a have been defined only for arcs $a \in A \cap A(\bar{\mathcal{G}}_r^R)$, i.e., all arcs covered by at least one good path in our set. By construction, the optimal path of the SPPCC-W instance is likely to be fully or mostly contained in this set. For that reason, we assign projected costs that are higher than average to the other arcs. We define $m_R := \max\limits_{a \in A(\bar{\mathcal{G}}_r^R)} \dfrac{x_a}{d_a}$. That is, m_R is the maximum cost-distance ratio in R. Then, for $a \in A \backslash A(\bar{\mathcal{G}}_r^R)$, we define $x_a := m_R \cdot d_a$.

A computational evaluation of the `AUC` approach is presented in Section 5.5.

5.4.2 Bound Underestimation: An LP-Based Cost Projection Method

In this section, we present linear programming (LP) based method for computing projected costs that we call `BOUND`. We use the following formulation:

$$\text{(LP-B)} \quad \begin{array}{lllll} \min & \sum_{a \in R} x_a & & & (5.7\text{a}) \\ \text{s.t.} & x(p) & \geq \eta \cdot \gamma_R(p) & \forall p \in \mathcal{G}_r^R & (5.7\text{b}) \\ & x_a/d_a & \geq \alpha & \forall a \in R & (5.7\text{c}) \\ & x_a/d_a & \leq \beta & \forall a \in R & (5.7\text{d}) \\ & \beta & \leq \tau \cdot \alpha & & (5.7\text{e}) \\ & \alpha\,,\, \beta & \geq 0 & & (5.7\text{f}) \end{array}$$

where $1 > \eta > 0$, $\tau > 0$ are parameters. The heart of (LP-B) is given by constraints (5.7b), which can be seen as a relaxation of equality (5.4). For a good path it ensures that the total projected costs are not allowed to underestimate the corresponding crossing costs by much. Combined with the objective function (5.7a), it forces crossing costs and corresponding projected costs to be close to each other. Constraints (5.7c)-(5.7f) are meant to keep unit costs within reasonable bounds, thus avoiding projected costs that are negative or very close to zero. Note that the number of constraints (5.7b) is exponential. However, we have the following:

Proposition 5.1. (LP-B) *can be solved in pseudo-polynomial time.*

Proof. Given $x \in \mathbb{R}^R_+, \alpha, \beta > 0$, our goal is to show that we can separate over all constraints of (LP-B) in pseudo-polynomial time. This reduces to separating over constraints (5.7b), since all others are polynomial in number. Given an entry-exit pair $(u, v) \in V^{\text{entry}}(R) \times V^{\text{exit}}(R)$, we know that the right-hand side of (5.7b) will be the same for all (u, v)-paths in R, say γ^R_{uv}. Thus, we would like to know whether there exists an (u, v)-path in $\mathcal{G}^R_r$ such that $\sum_{a \in p} x_a < \gamma^R_{uv}$. We recall that, by definition, $p \in \mathcal{G}^R_r$ if and only if $\sum_{a \in p} d_a \leq r \cdot d(u, v)$. It is thus clear that (since there is a polynomial number of entry-exit pairs), solving the separation problem is equivalent to solving an instance of the resource constrained shortest path problem (RCSPP) with costs x, resources d and resource limit $r \cdot d(u, v)$ for each entry-exit-pair (u, v) of a region. It is well-known [Has92] that the RCSPP can be solved in pseudo-polynomial time, thus completing the proof. $\square$

5.5 Computational Results

In this section, we evaluate the performance of the algorithms on real-world instances.

All algorithms were implemented in C++ and compiled with GCC 5.4.0. All computations were performed on machines with 132 GB of RAM and an Intel(R) Xeon(R) CPU E5-2660 v3 processor with 2.60GHz and 25.6 MB cache. The computation of projected costs was carried out in parallel using 28 threads. All other computations were carried out in single-thread mode. We used Gurobi 7.0.2 as our LP-solver.

5.5.1 Instances

All instances used in our computations correspond to real-world data, provided by Lufthansa Systems GmbH & Co. KG. This data comprises the airway network graph, the weather prognoses, and the set of overflight charges.

For our queries, we use a list of 4917 Origin-Destination (OD) pairs for which a commercial aircraft route exists and such that the optimal route does not leave the European airspace for any of the given weather prognoses. The reason for selecting these particular instances is that all airspaces involved define overflight fees using linear functions and the great-circle-distance (see Table 5.1), which is the variant that we considered in this paper.

The directed graph D corresponds to an horizontal layer of the airway network at $30,000$ feet altitude, which is a flight level common for cruising. It consists of $97,534$ nodes and $136,903$ arcs. Moreover, we consider three weather prognoses corresponding to January 6th 2015, November 21st 2016, and April 25th 2017. We identify them by the names *Jan*, *Nov*, and *Apr*, respectively. These prognoses contain the wind information defining the distortion factor ω_a for each arc $a \in D$ (see Figure 5.2). We use overflight charges and average fuel costs corresponding to an Airbus 320 aircraft. The fuel cost used is 1.73USD/km.

To gain some perspective on the size of the regions used for our computations, in Table 5.3 we describe a few properties of some of the instances used.

5.5.2 Computational Results

The two cost-projection methods presented rely on sets of good paths $\bar{\mathcal{G}}^R_r$ and $\mathcal{G}^R_r$, respectively. For comparability and computational efficiency, in this section we restrict both algorithms to use a single set $\bar{\mathcal{G}}^R_r$, which we define as follows for every region $R \in \mathcal{R}$.

Table 5.3: The first three columns show the number of arcs, entry- and exit nodes corresponding to a few European airspaces. The fourth column represents the cost per km used for the airspaces' overflight charges, i.e., the factor α_R defined in Section 5.3.

	entry nodes	exit nodes	arcs	cost per km
	(#)	(#)	(#)	(USD)
Germany	1938	1906	6361	1.58
Italy	1078	1258	7361	1.82
Norway	1705	1636	6863	1.08
Sweden	4922	4905	9561	1.39
USA	5689	5341	88782	0.22

We start with an empty set $\bar{\mathcal{G}}_r^R$. For every entry-exit pair (u, v) in R and for every $a \in R$, we compute the φ-shortest (u, v)-path in R that contains the arc a (easily obtainable by concatenating two φ-shortest paths and a), and insert it to $\bar{\mathcal{G}}_r^R$ only if it is cycle-free and it satisfies (5.3). Furthermore, for every (u, v) we store only the 10 shortest paths obtained in this way. For our experiments, we use $r = 1.2$.

We also set the parameters in (LP-B) to $\eta = 0.98$ and $\tau = 1.1$.[b]

While the LP models for all European airspaces could be solved in a reasonable time by Gurobi, and we only consider OD-pairs crossing these regions, the long-term goal is to use these methods worldwide. For that reason, we also compute projected costs for the USA airspace, which is particularly challenging, due to its size (see Table 5.3). In particular, even after restricting the set of good paths used as described above, the number of constraints (5.7b) exceeds 58 million. In conjunction with approximately $88,000$ variables, this leads to Gurobi quickly running out of memory. Inspired by the framework presented in Section 5.4.2, we implemented a rudimentary constraint generation algorithm which stores all constraints in memory, then iteratively adds subsets of those that are violated to the model and solves it, until no constraints are violated. This simple approach makes it possible to solve the LP model for USA. For completeness, we ran this constraint generation approach on all instances.

Figure 5.3 shows the preprocessing times needed to compute projected costs in each European region. As could be expected, the LP-based approach is always slower than the combinatorial `AUC` method. In our testing environment, we can handle the size of the generated models using the `BOUND` method. In nearly all cases, the latter is easier to solve in a single run than using constraint generation. However, for a couple of large regions (Norway and USA), the reverse is true.

Using constraint generation for the USA region enables us to solve the corresponding model. To do so, Gurobi requires 63.6GB of memory and it takes $8,696$s to find an optimal solution. Only 27.9 million constraints have to be added to the model, since all others never get violated.

Finally, in Table 5.4 we measure the quality and efficiency of our algorithms. Both cost-projection heuristics find solutions with a very small optimality gap. In particular, there is a large percentage of instances for which they find the optimal solution. Furthermore, the speedup of more than a factor of 350 with respect to 2LDA is remarkable.

[b]This seemingly deliberate selection of parameters yielded the best results among some tested variants. However, there is definitely plenty of room for further parameter optimization.

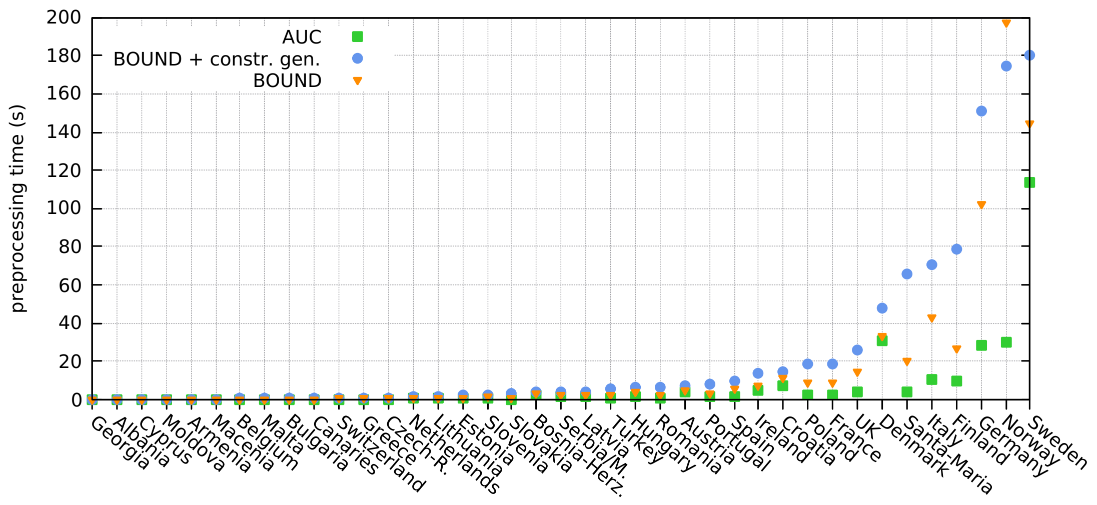


Figure 5.3: Time needed to compute projected costs using different methods on all airspaces

Table 5.4: For each weather prognosis, we present the time needed by 2LDA. For both heuristics, the average error, relative error, and speedup with respect to 2LDA is shown. Column *exact* lists the percentage of perfect hits, i.e., instances for which the error is zero.

	AUC				BOUND			
	abs err (USD)	rel err (%)	exact (%)	speedup (×)	abs err (USD)	rel err (%)	exact (%)	speedup (×)
Jan	34.84	1.03	35.27	382.63	4.83	0.16	36.89	380.00
Nov	34.60	1.03	34.29	362.15	4.73	0.16	36.81	363.48
Apr	35.44	1.06	35.57	361.63	4.70	0.16	37.60	364.44

5.6 Conclusions

In this paper, we introduced the SPPCC and the SPPCC-W, which models the flight trajectory optimization problem under influence of overflight fees and wind. We presented the Two-Layer-Dijkstra Algorithm for solving the SPPCC-W to optimality in polynomial time. We also developed a novel cost-projection method and suggested two computational procedures. In the corresponding computations, we showed that they achieve approximation errors well below 1%, while yielding a speedup of over 350 with respect to the Two-Layer-Dijkstra Algorithm.

5.7 Acknowledgements

We thank Lufthansa Systems GmbH & Co. KG for providing us with the data used in this paper, as well as for the many fruitful discussions.

References

[ASE16] ASECNA. *Air Navigation Services Charges.* [Online; accessed 8-April-2016]. 2016. URL: `http://www.ais-asecna.org/pdf/gen/gen-4-2/00gen4-2-01.pdf`.

[Bar+08] C. Barrett, K. Bisset, M. Holzer, G. Konjevod, M.V. Marathe, and D. Wagner. "Engineering label-constrained shortest-path algorithms". In: *Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management (AAIM'08).* Vol. 5034. Lecture Notes in Computer Science. Springer, June 2008, pp. 27–37.

[Bla+16c] Marco Blanco, Ralf Borndörfer, Nam Dung Hoang, Anton Kaier, Thomas Schlechte, and Swen Schlobach. *The Shortest Path Problem with Crossing Costs.* eng. Tech. rep. 16-70. Takustr.7, 14195 Berlin: ZIB, 2016.

[Bla+16d] Marco Blanco, Ralf Borndörfer, Nam Dũng Hoàng, Anton Kaier, Adam Schienle, Thomas Schlechte, and Swen Schlobach. "Solving Time Dependent Shortest Path Problems on Airway Networks Using Super-Optimal Wind". In: *16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2016).* Vol. 54. This work is licensed under the Creative Commons Attribution 3.0 Unported License. To view a copy of this license, visit `http://creativecommons.org/licenses/by/3.0/`. 2016. DOI: `10.4230/OASIcs.ATMOS.2016.12`.

[Bon+13b] Pierre Bonami, Alberto Olivares, Manuel Soler, and Ernesto Staffetti. "Multiphase Mixed-Integer Optimal Control Approach to Aircraft Trajectory Optimization". In: *Journal of Guidance, Control, and Dynamics* 36.5 (July 2013), pp. 1267–1277. ISSN: 0731-5090. DOI: `10.2514/1.60492`.

[Bro+05] Hajo Broersma, Xueliang Li, Gerhard Woeginger, and Shenggui Zhang. "Paths and cycles in colored graphs". In: *Australasian journal of combinatorics* 31 (2005), pp. 299–311.

[Car07] Susan Carey. "Calculating Costs in the Clouds". In: *The Wall Street Journal* (June 2007).

[EUR16a] EUROCONTROL. *Establishing Route Charges.* [Online; accessed 8-April-2016]. 2016. URL: `http://www.eurocontrol.int/articles/establishing-route-charges`.

[Fed16] Federal Aviation Administration. *Overflight Fees.* [Online; accessed 8-April-2016]. 2016. URL: `https://www.faa.gov/air_traffic/international_aviation/overflight_fees/`.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* New York, NY, USA: W. H. Freeman & Co., 1979. ISBN: 0716710447.

[Has92] Refael Hassin. "Approximation Schemes for the Restricted Shortest Path Problem". In: *Mathematics of Operations Research* 17.1 (1992), pp. 36–42.

[HMS07] Refael Hassin, Jérôme Monnot, and Danny Segev. "Approximation algorithms and hardness results for labeled connectivity problems". In: *Journal of Combinatorial Optimization* 14.4 (2007), pp. 437–453.

[HS04] Horst W. Hamacher and Anita Schöbel. "Design of Zone Tariff Systems in Public Transport". ngerman. In: *Operations Research* 52.6 (2004), pp. 897–908.

[ID05] Stefan Irnich and Guy Desaulniers. “Column Generation”. In: Boston, MA: Springer US, 2005. Chap. Shortest Path Problems with Resource Constraints, pp. 33–65. ISBN: 978-0-387-25486-9.

[Kar+12a] Stefan E. Karisch, Stephen S. Altus, Goran Stojković, and Mirela Stojković. “Operations”. English. In: *Quantitative Problem Solving Methods in the Airline Industry.* Ed. by Cynthia Barnhart and Barry Smith. Vol. 169. International Series in Operations Research & Management Science. Springer US, 2012, pp. 283–383. ISBN: 978-1-4614-1607-4.

[LR01] Dean H. Lorenz and Danny Raz. “A Simple Efficient Approximation Scheme for the Restricted Shortest Path Problem”. In: *Oper. Res. Lett.* 28.5 (June 2001), pp. 213–219. ISSN: 0167-6377.

[Som14] Christian Sommer. “Shortest-path Queries in Static Networks”. In: *ACM Computing Surveys* 46.4 (Mar. 2014), 45:1–45:31. ISSN: 0360-0300.

[SPS03] Robert L. Schultz, Stephen G. Pratt, and Donald A. Shaner. *Four-dimensional route planner.* WO Patent App. PCT/US2002/029,474. Mar. 2003.

[Sri+15] Banavar Sridhar, Hok Kwan Ng, Florian Linke, and Neil Y. Chen. “Impact of Airspace Charges on Transatlantic Aircraft Trajectories”. In: *15th AIAA Aviation Technology, Integration, and Operations Conference.* American Institute of Aeronautics and Astronautics, June 2015. DOI: `10.2514/6.2015-2596`. URL: `https://doi.org/10.2514/6.2015-2596`.

5.8 Appendix: The Two-Layer-Dijkstra Algorithm

Here, we give a detailed description of the Two-Layer-Dijkstra Algorithm presented in Section 5.3.

The complete version of the Two-Layer-Dijkstra Algorithm can be found in Algorithm 2. We use the same notation as in Section 5.3. In particular, we make use of the subroutine ComputeCost($\bar{a}$) defined before. We also define the subroutine ShortestPath(s, t, D, φ), which takes a directed graph, a pair of nodes in that digraph, and a non-negative real function on the arcs; and uses Dijkstra's algorithm to return the shortest path connecting the two nodes in the digraph.

Algorithm 2 Two-Layer-Dijkstra Algorithm for SPPCC

```
Input: $D$, $s, t$, $\varphi$, disjoint regions $\mathcal{R}$ and linear $f_R$ for $R \in \mathcal{R}$.
Output: $(s,t)$-path $p$ in $D$
 1: Construct $\bar{D}$, insert dummy node $v^{\text{null}}$ to $\bar{V}$          ▷ $\bar{D}$ as in Section 5.3
 2: for all $\bar{v} \in \bar{V}$ do
 3:     $dist[\bar{v}] \leftarrow \infty$                                  ▷ initialize distances
 4:     $pred[\bar{v}] \leftarrow v^{\text{null}}$                             ▷ initialize predecessors
 5: $dist[s] \leftarrow 0$
 6: $Q \leftarrow \bar{V}$
 7: while $Q \neq \emptyset$ do
 8:     $\bar{u} \leftarrow \text{argmin}_{\bar{v} \in Q}(dist[\bar{v}])$          ▷ choose node with minimum distance
 9:     if $\bar{u} = t$ then
10:         break
11:     $Q \leftarrow Q \backslash \{\bar{v}\}$
12:     for all $\bar{v}$ s.t. $(\bar{u}, \bar{v}) \in \bar{A}$ do
13:         $\ell_{\bar{u},\bar{v}} \leftarrow$ ComputeCost($(\bar{u}, \bar{v})$)    ▷ compute cost corresponding to arc $(\bar{u}, \bar{v})$
14:         if $dist[\bar{v}] > dist[\bar{u}] + \ell_{\bar{u},\bar{v}}$ then         ▷ if new cost is better
15:             $dist[\bar{v}] \leftarrow dist[\bar{u}] + \ell_{\bar{u},\bar{v}}$           ▷ update distance
16:             $pred[\bar{v}] \leftarrow \bar{u}$                          ▷ update predecessor
17: $p \leftarrow \emptyset, u \leftarrow t$
18: while $pred[u] \neq v^{\text{null}}$ do
19:     $p \leftarrow$ ShortestPath($pred[u], u, D, \varphi$) $\cup\, p$      ▷ reconstruct path in $D$ recursively
20: return $p$
```

Introduction to Chapter 6

This chapter was published in [Bla+15]. It is the only main chapter in this thesis that does not investigate algorithms for solving the Flight Planning Problem. Instead, it has a much heavier focus on theory than the other chapters.

We investigate the Path Avoiding Forbidden Pairs (PAFP) Problem, in which the goal is to find a path between two given nodes while ensuring that out of a given set of pairs of arcs, no pair is contained in the path. This problem is well known to be NP-hard [GMO76].

We study the polytope defined by its feasible solutions and provide a complete linear description for the case in which the graph is acyclic and the forbidden pairs satisfy a *disjointness* property. Despite the exponential number of facets, the separation problem can be solved in polynomial time. While the PAFP Problem has been thoroughly studied in the literature, this is the first polyhedral analysis of it.

Beyond the results themselves, a significant contribution of this research is the technique used to prove that the proposed inequalities are facet-defining. Since the dimension of the polytope possibly decreases as more pairs are added, it is not obvious how to follow the traditional approach of showing that the face defined by an inequality has dimension equal to that of the polytope minus one. Instead, we show that the points on the hyperplane defined by the inequality, together with any point in the polytope but outside the hyperplane, can generate any point in the polytope via an affine combination. Unfortunately, due to a constraint on the number of pages, this proof could not be included in the published article. We have added it in Appendix 6.4, together with other missing proofs. Note that this appendix is not part of the original publication.

The PAFP Problem models a particular case of the traffic restrictions present in the Flight Planning Problem, as illustrated in Section 1.1.4. This link is what pointed us in the direction of this research. Seeing as traffic restrictions in practice are much more general and the flight planning problem is usually not solved using Integer Programming, additional work is required to apply the results of this chapter to Flight Planning. For efficient algorithms that are able to handle traffic restrictions in practice, we refer the reader to [KCL17] and [KCL18].

References

[Bla+15] Marco Blanco, Ralf Borndörfer, Michael Brückner, Nam Dũng Hoàng, and Thomas Schlechte. "On the Path Avoiding Forbidden Pairs Polytope". In: *Electronic Notes in Discrete Mathematics* 50 (2015). LAGOS'15 - VIII Latin-American Algorithms, Graphs and Optimization Symposium. This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License. To view a copy of this license, visit `https://creativecommons.org/licenses/`

by-nc-nd/4.0/., pp. 343–348. ISSN: 1571-0653. DOI: 10.1016/j.endm.2015.07.057.

[GMO76] Harold N. Gabow, Shachindra N. Maheshwari, and Leon J. Osterweil. "On two problems in the generation of program test paths". In: *Software Engineering, IEEE Transactions on* 3 (1976), pp. 227–231.

[KCL17] Anders Nicolai Knudsen, Marco Chiarandini, and Kim S. Larsen. "Constraint Handling in Flight Planning". In: *Principles and Practice of Constraint Programming - 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*. 2017, pp. 354–369. DOI: 10.1007/978-3-319-66158-2_23. URL: https://doi.org/10.1007/978-3-319-66158-2_23.

[KCL18] Anders Nicolai Knudsen, Marco Chiarandini, and Kim S. Larsen. "Heuristic Variants of A* Search for 3D Flight Planning". In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 15th International Conference, CPAIOR 2018, Delft, The Netherlands, June 26-29, 2018, Proceedings*. 2018, pp. 361–376. DOI: 10.1007/978-3-319-93031-2_26. URL: https://doi.org/10.1007/978-3-319-93031-2_26.

Chapter 6

On the Path Avoiding Forbidden Pairs Polytope

Marco Blanco, Ralf Borndörfer, Michael Brückner, Nam Dũng Hoàng, Thomas Schlechte

Abstract

Given a directed, acyclic, complete graph, a source and a sink node, and a set of *forbidden* pairs of arcs, the *path avoiding forbidden pairs* (PAFP) problem is to find a path that connects the source and sink nodes and contains at most one arc from each forbidden pair. The general version of the problem is NP-hard, but it becomes polynomially solvable for certain topological configurations of the pairs. We present the first polyhedral study of the PAFP problem. We introduce a new family of valid inequalities for the PAFP polytope and show that they are sufficient to provide a complete linear description in the special case where the forbidden pairs satisfy a *disjointness* property. Furthermore, we show that the number of facets of the PAFP polytope is exponential in the size of the graph, even for the case of a single forbidden pair.

6.1 Introduction

Let $D = (V, A)$ be a directed, acyclic, complete graph with nodes $V = \{0, \dots, n\}$ labeled according to a topological order $\prec$, such that each arc $(u, v) \in A$ satisfies $u \prec v$. Let $\mathcal{F} \subseteq A \times A$ be a set of pairs of arcs. The *path avoiding forbidden pairs* (PAFP) problem is to find a $(0, n)$-path $\mathcal{P} \subseteq A$ in D such that $|\mathcal{P} \cap F| \leq 1$ for each $F \in \mathcal{F}$.

The equivalent problem that considers forbidden pairs of nodes first arose in the literature on automatic software testing [GMO76], where the execution flow through a large program is modeled in terms of a graph and forbidden pairs represent code segments which cannot both be executed in the same run. Further applications in bioinformatics include the problems of peptide sequencing [Che+01] and gene finding using RT-PCR tests [Kov13]. Our interest in the problem stems from an aircraft routing application, where feasible trajectories correspond to paths in an airway network. The routes must satisfy certain restrictions imposed by the air traffic service authorities, and these restrictions can be modeled in terms of forbidden pairs of arcs.

The PAFP problem in its decision version has long been known to be NP-hard [GMO76]. The optimization version, where the goal is to minimize the number of violated pairs, was recently shown to be NP-hard to approximate within a linear factor [Haj+10]. If the set of forbidden pairs has certain nesting properties , the problem can be solved in polynomial time by using contraction techniques [KP09] or dynamic programming [Kov13].

Some path problems with additional constraints, such as the hop-constrained path problem [Ste09], have been studied from a polyhedral point of view, leading to the identification of classes of facet-defining inequalities. A complete description has recently been derived for a related (minimum spanning tree) problem, in which the inclusion of a single pair of additional arcs incurs additional costs [BK14]. This description contains an exponential number of facet defining inequalities of subtour elimination type. To the best of our knowledge, the PAFP polytope has not been studied before.

In this article we present a complete linear description for the path avoiding a single arc pair (PASP) polytope using so called *$r\backslash\ell$-cut inequalities.* This result can be generalized to the case of a finite set of forbidden pairs satisfying a disjointness property. The number of facets of the PASP polytope is exponential, however, they can be separated in polynomial time. This polyhedral approach is useful in more complex problems involving additional constraints.

We use the following notation. Let the $(0, n)$-path polytope on D be denoted by P. It is well known that

$$P = \{x \in \mathbb{R}^A | Nx = e_0 - e_n, x \geq 0\},$$

where N is the node-arc incidece matrix of D and e_v is the v-th unit vector in $\mathbb{R}^n$.

Let $P(\mathcal{F})$ be the convex hull of all incidence vectors of $(0, n)$-paths in D avoiding forbidden pairs:

$$P(\mathcal{F}) := \operatorname{conv}\{x \in P \cap \mathbb{Z}^{|A|} \mid x_\ell + x_r \leq 1 \qquad \forall (\ell, r) \in \mathcal{F}\}.$$

Throughout this paper, we will use $x(C)$ to denote $\sum_{a \in C} x_a$ for $C \subseteq A$.

6.2 A complete linear description of the PASP polytope

In this section, we restrict ourselves to the case $|\mathcal{F}| = 1$. Suppose $\mathcal{F} = \{(\ell, r)\}$, with $\ell = (\ell^-, \ell^+)$ and $r = (r^-, r^+)$. Without loss of generality we can assume that $\ell^+ \preceq r^-$ and that there exists a path in D that uses both ℓ and r, since otherwise the forbidden pair constraint is redundant. Denote by V_r the set of nodes in V from which r^- can be reached (including r^- itself), by A_r the set of arcs with both ends in V_r, set $A_{r\backslash\ell} := A_r\backslash\{\ell\}$ and let $D_{r\backslash\ell} := (V_r, A_{r\backslash\ell})$ be a subgraph of D..

Let $\mathcal{C}_{r\backslash\ell}$ be the set of inclusion-minimal $(0, r^-)$-cuts in $D_{r\backslash\ell}$ and call such a cut an $r\backslash\ell$-cut. Consider the family of $r\backslash\ell$-cut inequalities:

$$x(C) - x_r \geq 0 \qquad\qquad \forall C \in \mathcal{C}_{r\backslash\ell} \tag{6.1}$$

Lemma 6.1. The $r\backslash\ell$-cut inequalities (6.1) are valid for $P(\mathcal{F})$.

We now show that these inequalities provide a complete linear description of $P(\mathcal{F})$.

Theorem 6.1. *If $|\mathcal{F}| = 1$, $P(\mathcal{F}) = \{x \in P | x(C) - x_r \geq 0 \quad \forall C \in \mathcal{C}_{r\backslash\ell}\}$.*

Proof. Let $Q = \{x \in P | x$ satisfies (6.1)$\}$. By Lemma 6.1 we know $P(\mathcal{F}) \subseteq Q$. It can be verified that $x_\ell + x_r \leq 1$ is the $r\backslash\ell$-inequality corresponding to $C' := \{(u, v) \in A_{r\backslash\ell} | u \leq \ell^-, \ell^+ \leq v\}$, so $P(\mathcal{F}) = \operatorname{conv}(Q \cap \mathbb{Z}^n)$. Thus, all we need to prove is that Q is an integral polytope.

We start with a useful observation: Every $\hat{x} \in Q\backslash P(\mathcal{F})$ satisfies $\hat{x}_\ell > 0$ and $\hat{x}_r > 0$. To see this, let us express $\hat{x}$ as a convex combination of vertices of P. Since $\hat{x} \notin P(\mathcal{F})$, there must be at least one vertex of P with $x_\ell = 1$ and $x_r = 1$ contributing to the convex combination. Since all points in P satisfy nonnegativity constraints, $\hat{x}$ must necessarily have $\hat{x}_\ell > 0$ and $\hat{x}_r > 0$.

Suppose Q has a fractional vertex x^*. By the previous observation, we have $x^*_r, x^*_\ell > 0$. We now show how to decompose x^* into two flows, one of them avoiding r and the other ℓ.

Let us consider the graph D_r obtained from D by deleting ℓ and each arc in A that does not lie on any $(0, n)$-path that also contains r. By definition, we do not delete any arcs on a (r^+, n)-path, and the only arc on a $(0, r^-)$-path we delete is ℓ. The subgraph of D_r induced by all nodes that can reach r^- is precisely $D_{r\setminus\ell}$. We define upper capacities on the arc set of D_r by $c_a := x^*_a$.

Now, we compute a maximum $(0, n)$-flow x^r on D_r. Since $\{r\}$ is clearly a cut with capacity x^*_r, we know that the value of x^r is upper-bounded by x^*_r.

We will now prove that x^r is lower-bounded by (and thus equal to) x^*_r. Consider a minimum-capacity $(0, n)$-cut C in D_r. By the classical max-flow-min-cut theorem, we only need to prove that it has capacity x^*_r. Without loss of generality, we can assume C is inclusion-minimal.

Case 1 C disconnects 0 from r^-. By definition of $\mathcal{C}_{r\setminus\ell}$, there must exist $C' \in \mathcal{C}_{r\setminus\ell}$ with $C' \subseteq C$. Since (6.1) are satisfied by x^* and $C' \subseteq A_{r\setminus\ell}$, we have

$$x^*_r \leq x^*(C') = c(C') \leq c(C).$$

Case 2 Case 1 does not hold and C disconnects r^+ from n. By minimality of C, its intersection with $A_{r\setminus\ell}$ is empty. Thus, all arcs in C belong to D_r and $c(C) = x^*(C)$. Since the (r^+, n)-flow induced by x^* is at least x^*_r, we have

$$x^*_r \leq x^*(C) = c(C).$$

Case 3 Case 1 and 2 do not hold and $C = \{r\}$. Since r belongs to D_r, we have

$$x^*_r = c_r = c(C).$$

Since the capacity of each $(0, n)$-cut is lower-bounded by x^*_r, the max-flow min-cut theorem implies that x^r has a value of at least x^*_r. Combining this with the upper bound, we deduce that the value of x^r is exactly x^*_r.

It is easy to see that $x^\ell := x^* - x^r$ satisfies the flow-conservation constraints and nonnegativity. That is, x^ℓ is a $(0, n)$-flow of value $1 - x^*_r$. Clearly, $x^\ell_r = 0$.

In other words, we can express x^* as the sum of two $(0, n)$-flows x^r and x^ℓ, such that $x^r_\ell = 0$ and $x^\ell_r = 0$.

Scaling the flows x^r and x^ℓ to value 1, we obtain $(0, n)$-flows $y^r := \frac{1}{x^*_r} x^r$ and $y^\ell := \frac{1}{1-x^*_r} x^\ell$ that are contained in $P(\mathcal{F})$, and can express x^* as

$$x^* = x^r + x^\ell = x^*_r y^r + (1 - x^*_r) y^\ell.$$

Since x^* is a convex combination of both, we have $x^* \in P(\mathcal{F})$, contradicting the initial assumption. □

The construction used in the last proof leads to:

Corollary 6.1. The system of inequalities $\{Nx = e_0 - e_n, x(C) - x_r \geq 0 \quad \forall C \in \mathcal{C}_{r\setminus\ell}, x \geq 0\}$ can be separated in polynomial time.

Proof. Since the number of flow-conservation constraints and bounds is polynomial, we can assume that $x^* \in P$, that is, x^* defines a $(0, n)$-flow of value 1.

Consider the graph D_r that we defined in the proof of Theorem 6.1, and use x^* as capacities for the arc variables. Let x' be a maximum $(0, n)$-flow over D_r satisfying the arc capacities. If the value of x' is equal to x^*_r, that means (by the max-flow min-cut theorem) that the capacity of every $(0, n)$-cut in D_r is at least x^*_r. As all cuts in $\mathcal{C}_{r\setminus\ell}$ are subsets of the arcs of D_r, all $r\setminus\ell$-cut-constraints (6.1) are satisfied. If the value of x' is smaller than x^*_r, then there must exist a bottleneck corresponding to a cut with capacity smaller than x^*_r and this cut must belong to $\mathcal{C}_{r\setminus\ell}$. We know we can identify the bottleneck in polynomial time, and thus a violated inequality in (6.1). □

Let Π be the set of infeasible paths, that is, $(0, n)$-paths in D that contain both ℓ and r. For each $\mathcal{P} \in \Pi$, consider the set of arcs in $A_{r\setminus\ell}$ such that their heads but not their tails are visited by $\mathcal{P}$ and are ordered between arcs ℓ and r: $C_{\mathcal{P},\ell,r} := A_{r\setminus\ell} \cap \delta^-(\{v \in V | \ell^+ \preceq v \preceq r^-, (u, v) \in \mathcal{P}$ for some $u\})$. For each $\mathcal{P} \in \Pi$, we have $C_{\mathcal{P},\ell,r} \in \mathcal{C}_{r,\ell}$. The following technical lemma shows the importance of inequalities corresponding to infeasible paths.

Lemma 6.2. For every $\mathcal{P} \in \Pi$, the inequality $x_r \leq x(C_{\mathcal{P},\ell,r})$ is facet defining. Furthermore, if $\mathcal{P}_1, \mathcal{P}_2 \in \Pi$ are not identical on the interval $[\ell^+, r^-]$, then their corresponding inequalities define different facets.

As an immediate consequence, we have:

Theorem 6.2. *The number of facets of $P(\mathcal{F})$ is exponential in $|V|$.*

6.3 The PAFP polytope for disjoint pairs

Finally, we extend our results to a more general case. Let $\mathcal{F} = \{(\ell_1, r_1), \ldots, (\ell_k, r_k)\}$, and let $\mathcal{F}_i := \{(\ell_i, r_i)\}$ for $i = 0, \ldots, n$. We say the forbidden pairs in $\mathcal{F}$ are *disjoint* if they satisfy the following:

$$0 \preceq \ell_1^- \prec \ell_1^+ \preceq r_1^- \prec r_1^+ \preceq \ldots \preceq \ell_k^- \prec \ell_k^+ \preceq r_k^- \prec r_k^+ \preceq n.$$

Using induction and the flow decomposition argument from the proof of Theorem 6.1, it follows:

Theorem 6.3. *Let $\mathcal{F}$ be a set of disjoint forbidden pairs. Then the following holds:*

$$P(\mathcal{F}) = \bigcap_{i=1}^{k} P^{\mathcal{F}_i}.$$

References

[BK14] Christoph Buchheim and Laura Klein. “Combinatorial optimization with one quadratic term: Spanning trees and forests”. In: *Discrete Applied Mathematics* 177.0 (2014), pp. 34–52. ISSN: 0166-218X. DOI: `http://dx.doi.org/10.1016/j.dam.2014.05.031`. URL: `http://www.sciencedirect.com/science/article/pii/S0166218X14002467`.

[Che+01] Ting Chen, Ming-Yang Kao, Matthew Tepel, John Rush, and George M Church. “A dynamic programming approach to de novo peptide sequencing via tandem mass spectrometry”. In: *Journal of Computational Biology* 8.3 (2001), pp. 325–337.

[GMO76] Harold N. Gabow, Shachindra N. Maheshwari, and Leon J. Osterweil. “On two problems in the generation of program test paths”. In: *Software Engineering, IEEE Transactions on* 3 (1976), pp. 227–231.

[Haj+10] MohammadTaghi Hajiaghayi, Rohit Khandekar, Guy Kortsarz, and Julián Mestre. "The checkpoint problem". In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques.* Springer, 2010, pp. 219–231.

[Kov13] Jakub Kováč. "Complexity of the path avoiding forbidden pairs problem revisited". In: *Discrete Applied Mathematics* 161.10-11 (2013), pp. 1506–1512. ISSN: 0166-218X. DOI: `10.1016/j.dam.2012.12.022`.

[KP09] Petr Kolman and Ondrej Pangrác. "On the complexity of paths avoiding forbidden pairs". In: *Discrete Applied Mathematics* 157.13 (2009), pp. 2871–2876. DOI: `10.1016/j.dam.2009.03.018`.

[Ste09] Rüdiger Stephan. "Facets of the (s,t)-p-path polytope". In: *Discrete Applied Mathematics* 157.14 (2009), pp. 3119–3132. ISSN: 0166-218X. DOI: `10.1016/j.dam.2009.06.003`.

6.4 Appendix: Missing Proofs

Proof. of Lemma 6.1. Let $x^{\mathcal{P}}$ be the incidence vector of a feasible path $\mathcal{P}$. If $r \notin \mathcal{P}$, then the inequalities are trivially satisfied by $x^{\mathcal{P}}$. Otherwise, by definition of A_r, we know every path that uses r must intersect every $(0, r^-)$-cut in A_r. Since feasible paths that use r do not use ℓ, they must intersect every $(0, r^-)$-cut in $A_{r\setminus\ell}$. □

Proof. of Lemma 6.2. First, we define some notation. We previously defined a path as a set of arcs. However, for this proof it will be useful to see it as a set of arcs and nodes. We say a set of arcs and nodes is a (u,w)-*split* if it is the union of two internally node-disjoint (u,v)-paths. For two paths $\mathcal{P}$ and $\mathcal{Q}$, we define $s(\mathcal{P},\mathcal{Q})$ as the number of splits that are contained in $\mathcal{P}\cup\mathcal{Q}$ between ℓ^+ and r^- and contain at least one node in $\mathcal{Q}\setminus\mathcal{P}$. We also define $[u,w] := \{v \in V | u \preceq v \preceq w\}$. For a path $\mathcal{P}$ containing nodes u and w, we define $\mathcal{P}[u,w]$ as its subpath from u to w.

Let $\mathcal{P} \in \Pi$. By definition of $C_{\mathcal{P},\ell,r}$, the vector corresponding to a path $\mathcal{P}^=$ satisfies (6.1) with equality if and only if one of the following holds:

1. $\mathcal{P}^=$ contains neither ℓ nor r and $\mathcal{P} \cap \mathcal{P}^= \cap [\ell^+, r^-] = \emptyset$.

2. $\mathcal{P}^=$ contains exactly one of ℓ and r, and $s(\mathcal{P},\mathcal{Q}) = 0$.

Since the dimension of $P(\mathcal{F})$ is unknown, we use the following proof technique: We show that for each feasible path $\mathcal{Q}$ that is not tight for the $r\setminus\ell$-inequality corresponding to $\mathcal{P}$, $x(\mathcal{P})$ can be expressed as

$$x(\mathcal{P}) = \lambda_{\mathcal{Q}} x_{\mathcal{Q}} + \sum_{i=1}^{m} \lambda_i x(\mathcal{P}_i),$$

where $\sum_{i=1}^m \lambda_i = 1$ and $\{P_i\}_{i=1}^m$ is the set of feasible paths that are tight for $x_r \leq x(C_{\mathcal{P},\ell,r})$. In particular, we will prove that if $|\mathcal{P} \cap \{\ell, r\}| = 1$ then $\lambda_{\mathcal{Q}} = -\frac{1}{s(\mathcal{P},\mathcal{Q})}$, while if $|\mathcal{P} \cap \{\ell, r\}| = 0$ then $\lambda_{\mathcal{Q}} = -\frac{1}{s(\mathcal{P},\mathcal{Q})+1}$.

If we prove this for each $\mathcal{Q}$, it follows that $P(\mathcal{F}) \subseteq \operatorname{aff}(\{x(\mathcal{P})\} \cup \{x(P_i)\}_{i=1}^m)$, and thus the corresponding $r\setminus\ell$-inequality defines a $(d-1)$-dimensional face, where d is the dimension of $P(\mathcal{F})$.

We will prove the claim by induction on $s(\mathcal{P},\mathcal{Q})$. For the basis step, we again distinguish multiple cases. In each case, we express $x(\mathcal{P})$ as an affine combination of $x(\mathcal{Q})$ and tight characteristic vectors of feasible paths.

1. $\mathcal{Q}$ contains neither ℓ nor r, and $s(\mathcal{P},\mathcal{Q}) = 0$.

   Let u be the smallest node in $[\ell^+, r^-]$ that $\mathcal{P}$ and $calQ$ have in common.

$$\begin{aligned} x(\mathcal{P}) &= (x^{\mathcal{P}[0,u]} + x^{\mathcal{Q}[u,n]}) \\ &\quad + (x^{\mathcal{Q}[0,u]} + x^{\mathcal{P}[u,n]}) \\ &\quad - x(\mathcal{Q}). \end{aligned}$$

2. $\mathcal{Q}$ contains neither ℓ nor r, and $s(\mathcal{P},\mathcal{Q}) = 1$.

   In this and in the subsequent cases, let S be the given (u,w)-split and let v be a node in

$(\mathcal{Q} \backslash \mathcal{P}) \cap S$.

$$\begin{aligned}
x(\mathcal{P}) = &\frac{1}{2}(x^{\mathcal{P}[0,u]} + x^{\mathcal{Q}[u,v]} + x_{(v,n)}) \\
&+ \frac{1}{2}(x_{(0,v)} + x^{\mathcal{Q}[v,w]} + x^{\mathcal{P}[w,n]}) \\
&- \frac{1}{2}(x_{(0,v)} + x_{(v,n)}) \\
&+ \frac{1}{2}(x^{\mathcal{Q}[0,u]} + x^{\mathcal{P}[u,n]}) \\
&+ \frac{1}{2}(x^{\mathcal{P}[0,w]} + x^{\mathcal{Q}[w,n]}) \\
&- \frac{1}{2}x(\mathcal{Q})
\end{aligned}$$

3. $\mathcal{Q}$ contains ℓ but not r, and $s(\mathcal{P}, \mathcal{Q}) = 1$.

$$\begin{aligned}
x(\mathcal{P}) = &(x^{\mathcal{Q}[0,v]} + x_{(v,n)}) \\
&+ (x_{(0,v)} + x^{\mathcal{Q}[v,w]} + x^{\mathcal{P}[w,n]}) \\
&+ x^{\mathcal{P}[0,w]} + x^{\mathcal{Q}[w,n]}) \\
&- (x_{(0,v)} + x_{(v,n)}) \\
&- x(\mathcal{Q})
\end{aligned}$$

4. $\mathcal{Q}$ contains r but not ℓ, and $s(\mathcal{P}, \mathcal{Q}) = 1$.

$$\begin{aligned}
x(\mathcal{P}) = &(x^{\mathcal{Q}[0,u]} + x^{\mathcal{P}[u,n]}) \\
&+ (x^{\mathcal{P}[0,u]} + x^{\mathcal{Q}[u,v]} + x_{(v,n)}) \\
&+ (x_{(0,v)} + x^{\mathcal{Q}[v,n]}) \\
&- (x_{(0,v)} + x_{(v,n)}) \\
&- x(\mathcal{Q})
\end{aligned}$$

Here, we used the fact that D is a complete digraph to guarantee the existence of $(0, v)$ and (v, n).

For the inductive step, we consider similar cases as above. In all cases, we assume that there exist at least two splits S fully contained in $\mathcal{P} \cup \mathcal{P}^{=}$ between ℓ^{+} and r^{-} that intersect $(\mathcal{Q} \backslash \mathcal{P})$. Let the first of these splits be defined by nodes u, v and w, as above.

1. $\mathcal{Q}$ contains neither ℓ nor r.

   By the induction hypothesis, we have

$$x(\mathcal{P}) = -\frac{1}{s(\mathcal{P}, \mathcal{Q})}(x_{(0,v)} + x^{\mathcal{Q}[v,n]}) + \sum_{i=1}^{m} \lambda_i x(\mathcal{P}_i),$$

with $\sum_{i=1}^{m} \lambda_i = 1 + \frac{1}{s(\mathcal{P},\mathcal{Q})}$. Now,

$$\begin{aligned}
&\frac{1}{s(\mathcal{P},\mathcal{Q})+1}(x^{\mathcal{P}[0,u]} + x^{\mathcal{Q}[u,v]} + x_{(v,n)}) \\
&+ \frac{1}{s(\mathcal{P},\mathcal{Q})+1}(x^{\mathcal{Q}[0,u]} + x^{\mathcal{P}[u,n]}) \\
&- \frac{1}{s(\mathcal{P},\mathcal{Q})+1}(x_{(0,v)} + x_{(v,n)}) \\
&- \frac{1}{s(\mathcal{P},\mathcal{Q})+1}x(\mathcal{Q}) \\
&+ \frac{s(\mathcal{P},\mathcal{Q})}{s(\mathcal{P},\mathcal{Q})+1}\sum_{i=1}^{m}\lambda_j x(\mathcal{P}_i) \\
=&\frac{1}{s(\mathcal{P},\mathcal{Q})+1}(x_{(0,v)} + x^{\mathcal{Q}[v,n]}) \\
&+ \frac{s(\mathcal{P},\mathcal{Q})}{s(\mathcal{P},\mathcal{Q})+1}\sum_{i=1}^{m}\lambda_j x(\mathcal{P}_i) \\
&+ \frac{1}{s(\mathcal{P},\mathcal{Q})+1}x(\mathcal{P}) \\
=&x(\mathcal{P})
\end{aligned}$$

2. $\mathcal{Q}$ contains ℓ but not r.
3. $\mathcal{Q}$ contains r but not ℓ.

By an analogous argument, we can deal with cases (ii) and (iii).

Finally, we need to prove that these facets are pairwise distinct. Let $\mathcal{P}$ and $\mathcal{Q}$ be two forbidden paths such that $\mathcal{P}[\ell^+, r^-]$ and $\mathcal{Q}[\ell^+, r^-]$ are not identical. Without loss of generality, suppose $s(\mathcal{P},\mathcal{Q}) \geq 1$, and define u, v, w as before. Again using the fact that the graph is complete, define the paths $\mathcal{P}' := ((0,v),(v,n))$ and $\mathcal{Q}' := ((0,v), x^{\mathcal{Q}[v,n]})$. It is not hard to see that $x(\mathcal{P}')$ is tight for $x_r \leq x(C_{\mathcal{P},\ell,r})$ and not for $x_r \leq x(C_{\mathcal{Q},\ell,r})$; while for $x(\mathcal{Q}')$ precisely the opposite holds. □

Proof. of Theorem 6.3. A great advantage of the proof in Section 6.2 is that Lemma 6.1 can be easily extended to the more general case of multiple forbidden pairs: Lemma 6.1 guarantees that all feasible path vertices are kept in the new polytope $P^{\mathcal{F}}$ and we know that all invalid ones are eliminated by the $r\backslash\ell$-inequalities. This holds independently of other constraints in $\mathcal{F}$. So again, the only point left to prove is the following lemma. □

Lemma 6.3. The polytope $\bigcap_{i=1}^{k} P^{\mathcal{F}_i}$ is integral.

Proof. Our proof takes an inductive approach over the number of considered forbidden pairs. Let $\mathcal{F}_{\leq i} := \bigcup_{j=1}^{i} \mathcal{F}_i$. The induction hypothesis is: $P^{\mathcal{F}_{\leq i-1}} \cap P^{\mathcal{F}_i} = P^{\mathcal{F}_{\leq i}}$ and $P^{\mathcal{F}_{\leq i}}$ is integral. The induction basis is already given in Section 6.2. Let us now solve the induction step:

The direction $P^{\mathcal{F}_{\leq i-1}} \cap P^{\mathcal{F}_i} \supseteq P^{\mathcal{F}_{\leq i}}$ directly follows the definition. For the converse, let us take a fractional vertex x in $P^{\mathcal{F}_{\leq i-1}} \cap P^{\mathcal{F}_i}$ and contradict its existence. We are going to see that x is contained in $P^{\mathcal{F}_{\leq i}}$ by expressing it as a convex combination of points in $P^{\mathcal{F}_{\leq i}}$.

By using the split method from Section 6.2, we can split the flow x into two flows $x = x^{\ell_i} + x^{r_i}$, where x^{r_i} contains the whole flow of x using arc r_i and x^{ℓ_i} is the remaining flow, containing also flow avoiding both ℓ_i and r_i.

Consider now the set $C := \{(u,v) \in A \mid u \preceq \ell_i^- \prec v\}$. Clearly, this set is an arc cut of D and since every arc is either jumping over ℓ_i^- or starting in ℓ_i^-, no path can intersect C twice. Let us consider now the path decompositions of the three points x, x^{ℓ_i}, x^{r_i} and call them M, M^{ℓ_i} and M^{r_i} respectively. Clearly, $M^{\ell_i} \cup M^{r_i}$ is a path decomposition of x as well, but in general a different one than M.

Let us now take a closer look at C. This cut separates the first $i-1$ forbidden pairs from the i-th one. M is a path decomposition, where all paths respect the first $i-1$ forbidden pairs, and $M^{\ell_i} \cup M^{r_i}$ a path decomposition respecting the i-th one. And clearly, both decompositions use C in the same amounts per arc. Since all the decompositions are clearly finite, we can compute a new decomposition of x by *mixing* the *left* part of M with the *right* part of $M^{\ell_i} \cup M^{r_i}$: For every arc $a \in C$, consider all paths in M (and $M^{\ell_i} \cup M^{r_i}$) using a. By arranging new paths out of the 0-a^--subpaths of the paths in M, the arc a itself and the a^+-n-subpaths contained in $M^{\ell_i} \cup M^{r_i}$, we get our new decomposition, called M^*. The paths in M^* respect all forbidden pairs in $\mathcal{F}^{\leq i}$ and hence, x is a convex combination of these valid paths. That is, x is contained in $P^{\mathcal{F}_{\leq i}}$. Since x was fractional, M contained at least 2 paths and hence, x is no vertex, which is our desired contradiction. $\square$

Chapter 7

Conclusion

We have presented multiple search algorithms and preprocessing techniques to find high-quality solutions for the Flight Planning Problem. Our algorithms are either the first of their kind or superior to the state-of-the-art. We have paid particular attention to the performance needed for consideration in real-world applications.

The Super Optimal Wind construction in Chapter 2 enabled us to design an A* algorithm for the case of flying at constant altitude. This algorithm is optimal under assumption of the FIFO property. It is on average 20 times faster than a time-dependent Dijkstra and needs less than 10s for its preprocessing phase. We also showed that a version of this algorithm, modified to run on a classical Time-Dependent Shortest Path Problem, is clearly superior to Time Dependent Contraction Hierarchies, the state-of-the-art approach for time-dependent route planning on road networks. The latter struggles not only with preprocessing times of several hours, but its queries are 15 times slower than those of the A*.

We reused the time underestimation techniques from that chapter to investigate a more general setting of the Flight Planning Problem in Chapter 3. On top of the assumptions considered in Chapter 2, this includes overflight charges, fuel consumption, and a variable flight altitude. Our pruning techniques were able to reduce the size of the graph by a factor of 10 and the runtime of a Dijkstra algorithm by a factor of 6. This is the first published work that shows how to consider all three cost components (fuel, time, and overflight charges) efficiently on real-world data.

Chapter 4 generalized the algorithm from Chapter 2 to run on a three-dimensional graph and again makes use of the Super Optimal Wind for time underestimation. The key contribution is a potential function based on simulating an idealized vertical profile of a flight over a given distance. This allows us to reliably underestimate fuel consumption. Under reasonable assumptions, this potential function is both admissible and consistent, thus making the resulting A* algorithm optimal. Thanks to sophisticated pre-processing steps, we finally obtain a heuristic that outperforms the current state-of-the-art. Indeed, we were able to demonstrate a speed-up in the query time of up to 40% and a smaller optimality gap.

In Chapter 5 we focused on the Shortest Path Problem with Crossing Costs, which models a basic version of the Flight Planning Problem under consideration of the common but complex Great Circle Distance-based overflight cost model. We introduced the only known polynomial time algorithm for the exact solution of this problem. Our Projected Costs technique leads to a heuristic that is 350 times faster than the exact approach, while preserving an optimality gap of less than 1%.

Chapter 6 is focused on pure mathematical research, motivated by the study of the Flight

Planning Problem. In it, we provided an in-depth polyhedral analysis of the Path Avoiding Forbidden Pairs (PAFP) polytope. We introduced a family of inequalities that provide a complete linear description of the polytope given a certain disjointness property, and which can be separated in polynomial time. Notably, we used a novel and ingenious technique for proving that each of these inequalities is facet-defining.

The techniques in Chapters 2 through 5 are extremely modular and can be combined in integrated algorithms to handle more advanced versions of the Flight Planning Problem. This is demonstrated by the use of the Super Optimal Wind from Chapter 2 in Chapters 3 and 4, as well as by the integration of the Two-Layer-Dijkstra Algorithm from Chapter 5 in Chapter 3.

While the findings presented in Chapter 6 may be of primary interest to mathematicians, they also offer potential benefits for applications in Flight Planning algorithms. By uncovering key structural insights, this research could ultimately pave the way for more efficient and effective flight planning strategies.

In summary, the results in this thesis establish a new standard in optimization algorithms for the Flight Planning Problem. They are suitable for use in practical applications and should be able to significantly improve existing systems. Furthermore, our approach and techniques are innovative and valuable from a mathematical perspective, providing insights that extend beyond the realm of aviation. These contributions position our work as a significant advancement in the field of optimization and pave the way for future research in this area.

Zusammenfassung

Das Flugplanungsproblem ist eine Art Kürzeste-Wege-Problem, das sich mit der Berechnung der kostengünstigsten Flugbahn zwischen zwei Flughäfen befasst. Diese Flugbahn muss verschiedene Betriebsbeschränkungen berücksichtigen. Weitere Faktoren wie Wind, Flugzeugleistung und der Treibstoff an Bord müssen berücksichtigt werden. Ziel ist es, die Summe aus Treibstoffkosten, Zeitkosten und Überflugkosten zu minimieren.

Trotz der Komplexität dieses Problems, das die Verarbeitung großer Datenmengen erfordert, erfordern reale Anwendungen eine Lösung innerhalb von Minuten oder sogar Sekunden. Daraus ergibt sich der Bedarf an schnellen Lösungsalgorithmen, die im Idealfall eine garantierte Optimalität aufweisen.

In dieser Arbeit untersuchen wir verschiedene Aspekte des Flugplanungsproblems und schlagen mehrere neuartige Algorithmen vor, die an realen Daten getestet wurden.

Wir stellen eine Methode zur zuverlässigen Unterschätzung der Luftentfernung vor, die es ermöglicht, einen A*-Algorithmus zur Lösung einer Variante des Flugplanungsproblems zu entwickeln, die von einer konstanten Höhe ausgeht. Dieser Algorithmus ist unter der Annahme der First-In-First-Out-Eigenschaft garantiert optimal und ist 20 Mal schneller als der Dijkstra-Algorithmus. Bei einem zeitabhängigen Kürzeste-Wege-Problem mit stückweise linearen Reisezeitfunktionen übertrifft unser A* den Stand der Technik um den Faktor 15.

Wir schlagen auch eine Pruning-Technik vor, um den Suchraum des Problems zu reduzieren und gleichzeitig die Optimalität zu erhalten. Wir zeigen, dass ein Dijkstra-Algorithmus, der auf diesem reduzierten Raum läuft, bis zu sechsmal schneller ist als einer, der auf einer naiven Suchraumreduktion basiert. Diese ist die erste Arbeit in der Literatur, die einen hochleistungsfähigen Algorithmus zur Lösung des dreidimensionalen Flugplanungsproblems unter Berücksichtigung von Treibstoff, Zeit, und Überflugkosten vorstellt.

Wir erweitern außerdem unseren ersten A*-Algorithmus zu einer dreidimensionalen Version, indem wir eine Potentialfunktion definieren, die auf idealisierten vertikalen Profilen basiert. Dieses Potenzial erweist sich unter vernünftigen Annahmen als zulässig und konsistent, so dass Optimalität gewährleistet ist. Darüber hinaus führen wir zwei Preprocessing-Techniken ein, um die Berechnung der Abfragen zu beschleunigen. Wir zeigen, dass unser Algorithmus gegenüber dem bisher besten A*-Algorithmus ein Speed-up von bis zu 40% erzielt. Darüber hinaus konzentrieren wir uns auf die Bedeutung von Überflugkosten bei der Flugplanung, die in der Literatur oft übersehen werden. Wir führen eine Heuristik ein, die streckenbasierte Kosten in bogenbasierte Kosten umwandelt und in einen klassischen Dijkstra-Algorithmus integriert. Der erforderliche Preprocessing-Schritt ist schnell genug, um in der Praxis eingesetzt werden zu können. Ein Dijkstra-Algorithmus, der die berechneten Bogenkosten verwendet, ist um einen Faktor von über 350 schneller als der beste bekannte exakte Algorithmus, mit einer Optimalitätslücke von weniger als 1%.

Schließlich führen wir eine polyedrische Analyse des Path Avoiding Forbidden Pairs Polytops durch, motiviert durch die Untersuchung komplexer Verkehrsbeschränkungen im Flugplanungsproblem. Unter bestimmten Bedingungen sind wir in der Lage, eine vollständige lineare Beschreibung des Polytops zusammen mit einer Trennungsroutine in Polynomialzeit zu liefern.

Selbstständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und alle dabei verwendeten Hilfsmittel und Quellen angegeben habe. Geistiges Eigentum anderer Autoren wurde als entsprechend gekennzeichnet. Ebenso versichere ich, dass ich an keiner anderen Stelle ein Prüfungsverfahren beantragt bzw. die Dissertation in dieser oder anderer Form an keiner anderen Fakultät als Dissertation vorgelegt habe.

Berlin, 14. März 2023,

Marco Blanco Sandoval