Freie Universität Berlin

Master's Thesis

Shortest Paths on Airway Networks

Author:
Adam Schienle

Referees:
Prof. Dr. Ralf Borndörfer
Prof. Dr. Alexander Bockmayr

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

Department of Mathematics

02 August 2016

Abstract

We study the Horizontal Flight Trajectory Optimisation Problem (HFTOP), where one has to find a cost-minimal aircraft trajectory between to airports s, t on the Airway Network, a directed graph. To this end, we distinguish three cases: a static one where no wind is blowing, and two cases where we regard the wind as a function of time. This allows us to model HFTOP as a Shortest Path Problem in the first case and a Time-Dependent Shortest Path Problem in the latter cases. While both of these problems are well-studied on road networks, the Airway Network we use was hitherto little considered in the literature.

In the static case, we compare the runtimes of Dijkstra's algorithm to those of A* and Contraction Hierarchies (CHs), the latter one being a state-of-the-art routing algorithm on road networks. We show that A* guided by a great-circle-distance potential yields speedups competitive to those of CHs.

In the time-dependent version, we study two different modelling approaches. Firstly, we compute the exact costs for the time on the arcs. In a second version, we use piecewise linear functions as the travel time. For both versions, we establish a criterion by which to check whether a given instance of HFTOP satisfies the FIFO property. Furthermore, we design problem-specific potential functions for the A* algorithm in both the PWL and the exact case. As the exact costs are non-linear, we introduce the notion of Super-Optimal Wind to underestimate the travel time on the arcs, and show that the Super-Optimal Wind yields a good underestimation in theory and an excellent approximation in practice. Moreover, we compare the runtimes of the time dependent versions of Dijkstra's Algorithm, A* and Time-dependent Contraction Hierarchies (TCHs) in the PWL case, showing that A* outperforms Dijkstra's Algorithm by a factor of 25 and TCHs by a factor of more than 16.

For the exact case, we compare Dijkstra's Algorithm to A* and show that using Super-Optimal Wind to guide the search leads to an average speedup of $\times 20$. We also computationally assess the error of the PWL approximation with respect to the exact solution.

Eigenständigkeitserklärung

Hiermit erkläre ich, nachfolgende Arbeit eigenständig und ohne Hilfe Dritter angefertigt zu haben. Alle Übernahmen aus der Literatur sind als solche gekennzeichnet und im Literaturverzeichnis aufgelistet.

Hereby I confirm that the work contained in this thesis is my own unless otherwise stated. All adoptions of literature have been referenced as such and are listed in the References section.

Berlin, 02 August 2016

Adam Schienle

Acknowledgements

This thesis arose from the project "Flight Trajectory Optimization on Airway Networks" at Zuse-Institute Berlin. I would like to thank the project team, in particular Prof. Dr. Ralf Borndörfer, Dr. Nam Dũng Hoàng and Marco Blanco for giving me the opportunity to work in the project and supervising my thesis. Lufthansa Systems supported the project, and thus, this work, for which I am grateful.

Thanks go to my family and friends for their continued support during writing this thesis.

Contents

1 Introduction and Basics **1**
- 1.1 Introduction . . . 1
- 1.2 Reviewing the Flight Trajectory Optimisation Problem . . . 3
- 1.3 Shortest Path Problem: A Review of Algorithmic Approaches . . . 3
- 1.4 Our Contributions . . . 5
- 1.5 Outline . . . 6
- 1.6 The Notational Ground . . . 6

2 The Horizontal Flight Trajectory Optimisation Problem **9**
- 2.1 An Aeronautics Primer . . . 9
- 2.2 The Flight Trajectory Optimisation Problem . . . 9
- 2.3 The Horizontal Flight Trajectory Optimisation Problem . . . 11
  - 2.3.1 Modelling HFTOP . . . 12
  - 2.3.2 How To: Obtain a TTF . . . 13

3 Algorithms for the Shortest Path Problem **17**
- 3.1 Dijkstra's Algorithm . . . 17
  - 3.1.1 The Static Case . . . 17
  - 3.1.2 Bidirectional Dijsktra . . . 20
  - 3.1.3 The Time-Dependent Case . . . 20
- 3.2 A* . . . 23
  - 3.2.1 A* in the Time-Dependent Case . . . 27
- 3.3 Contraction Hierarchies . . . 28
  - 3.3.1 Contraction Hierarchies in the Static Case . . . 28
  - 3.3.2 Contraction Hierarchies in the Time-Dependent Case . . . 33

4 Algorithms for HFTOP **35**
- 4.1 The Static Case . . . 35
- 4.2 The Exact Time Dependent Case . . . 36
  - 4.2.1 The Super-Optimal Wind Potential Function . . . 41
  - 4.2.2 Minimising the Crosswind and Maximising the Trackwind . . . 47
  - 4.2.3 Super-Optimal Wind in Practice . . . 51
- 4.3 Approximating the Travel Time Function . . . 51

5 Computational Results **55**
5.1 Instances 55
5.2 Results in the Static Case 56
5.3 Results in the PWL Case 57
5.4 Results in the Exact Case 58

6 Conclusion **62**

Bibliography **63**

1 Introduction and Basics

1.1 Introduction

With passenger air travel rising from year to year[Int16b], and the high fuel burn linked with it, reducing fuel consumption is one of the goals in the aviation industry. While this can in part be achieved through bigger aircraft or revised design, we will in this thesis focus on another important factor, namely the actual route an aircraft takes.

Planning a route is part of the process of flying: a dispatcher computes and submits a route to Air Traffic Control (ATC), who then accept or reject it. If accepted, the pilots must adhere to the planned route, and any deviations need to be approved by ATC. Commonly, a route is planned a few hours before the flight takes place, and takes into consideration many factors, the most important one being the fuel consumption en route. According to the Air Transport Action Group ATAG[Air16], the aviation industry burns around 1.5 billion barrels of Jet-A fuel per year, corresponding to 238.5 trillion litres, or roughly 82.4 billion USD[Int16a]. Even saving just 0.5% would add up to 1.2 billion litres (or 411.8 million USD) annually. Not only on a global scale, but also on an airline scale, savings can have a visible impact: according to Lufthansa[Luf15], their total fuel consumption in 2014 amounted to almost 8.9 million tons (6.9 billion litres). If one saves 0.5%, this means 19.2 million USD (35 million litres) per year, which yields not only a financial benefit for the airline involved, but also reduces the impact of aviation on the environment. The latter becomes apparent when considering the CO_2 emissions published by Lufthansa[Luf15]: their annual CO_2 emissions amount to 27.8 million tons. Saving 0.5% corresponds to 139 000 tons less CO_2 being released into the atmosphere.

One of the main problems in flight planning is the influence of wind. Aircraft typically fly at altitudes around FL330 ($\approx$ 10km), with strong high altitude winds blowing. These winds are taken into the extreme in jet streams, high altitude circum-polar winds always blowing in easterly direction (on the northern hemisphere). Their effect is noticeable to anybody flying for example from London to New York and back: the return trip may be up to two hours shorter than the outbound flight. The reason for this is that in the latter case, the aircraft circumnavigates the strong headwinds, while in the former one, it can travel at ease on the jet stream[Cri15]. But not only jet streams, even ordinary low pressure areas can create powerful winds in some altitudes, which aircraft can either use to their advantage or which they have to circumnavigate to avoid delay or, indeed, passenger discomfort. Two different minimum time trajectories, one without wind, the other considering it, can be found in Figure 1.1.

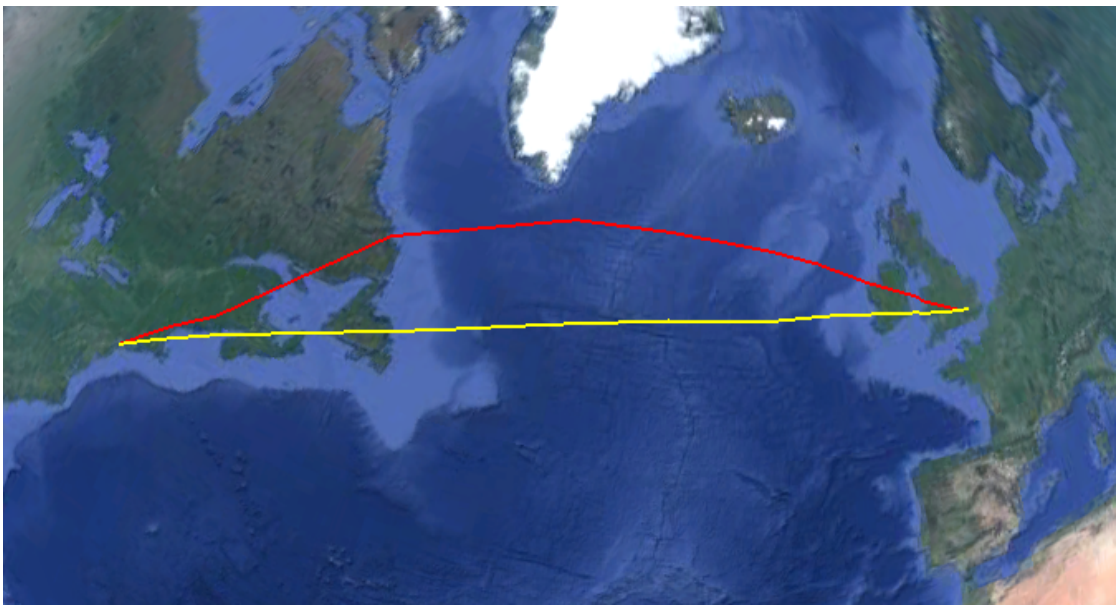


Figure 1.1: Two trajectories for the route from London to New York City. The static route is shown in yellow, the wind-dependent route in red (Image data: Google Earth, Digital Globe).

Intuitively, aircraft fly faster and more efficiently when being pushed along by tailwinds, while crosswinds hamper an aircraft's flight. For that reason, one tries to choose routes which exhibit these favourable conditions, and at the same time avoid routes with strong headwinds – all the while aiming to find the most efficient route.

For passengers, another aspect is important: if possible, the flight time should be as short as possible. In the way we model the problem, less fuel consumption also means shorter flight times, thus incorporating both aspects.

Nevertheless, overcrowding of airspaces may cause ATC to reject any submitted flight, as may inclement weather on the route. The avoidance of certain hazardous areas only renders the problem more difficult; but since reliable weather prognoses are only available a few hours in advance and ATC have to approve of any scheduled flight, there often is little time for planning. It is therefore paramount that flight planning takes as little time as possible while taking account of all constraints. Mathematically, we model flight planning as a shortest path problem (see the following chapter), which then needs to be solved efficiently – in particular, the actual query for the shortest path should be as fast as possible. This is the main goal of thesis at hand: we look into search algorithms for the shortest path problem, which solve it optimally and in as little time as possible. While these have been extensively studied for road networks, to the best of our knowledge there is no work available for aircraft routing in this particular setting.

1.2 Reviewing the Flight Trajectory Optimisation Problem

The scientific literature on flight planning approaches is scarce. One of the earliest approaches dates back to 1974's technical report by de Jong [dJ74], in which he first describes a 2D solution to the problem through various methods in continuous optimisation, assuming that free flight between any two points on the earth is possible. He then goes on to describe a graph which results from restricting traffic in some regions, and compares both approaches to each other. He continues by considering a 3D graph with instantaneous climb and descent, however, nodes in this graph are grouped, and edges only exist between these groups.

In [BOSS13], Bonami et al describe a multiphase mixed-integer approach to the problem, in which they combine optimal control and mixed-integer non-linear programming. The actual route, defined by waypoints, is described by integer variables, whereas external conditions such as wind are modelled through continuous variables. The authors study the case of a route from New York to Rome, using a restricted set of waypoints and a heuristic to solve it.

Modern aviation uses a very strict regulatory system where aircraft have to adhere to certain roads similar to the ones described by de Jong, or Bonami et al. This so-called Airway Network is for instance defined in [KASS12]. Here, the authors Karisch et al model the Flight Trajectory Optimisation Problem on this network, and propose a graph-theoretical solution. They consider a full 3D model of the Airway Network, and let the aircraft's speed constitute another dimension, thus yielding a 4D graph theoretical optimisation approach. As the full 4D approach is difficult to solve to optimality, they further consider a 2D+2D approach, where in a first step the aircraft's trajectory is optimised horizontally, and in a second step, the vertical profile and speed are optimised. We shall follow their idea, and use the *real-world* Airway Network as provided to us by Lufthansa Systems.

1.3 Shortest Path Problem: A Review of Algorithmic Approaches

The Shortest Path Problem (SPP) is one of the best-known problems in discrete optimisation. Given a graph with weighted arcs and two nodes, one seeks to find a path between these nodes such that the sum of the arc weights along the path is minimal. Its widespread applications range from traffic routing to social networks and even computer games, rendering it a well-studied problem. Consequently, there are many algorithms available for SPP – for a comprehensive survey, see [BDG$^+$15].

The arguably most important algorithm dates back to 1959, when E. Dijkstra developed his famous algorithm for shortest paths (see [Dij59]). Under mild assumptions, Dijkstra's algorithm solves the SPP in polynomial time, but in practice, it does not scale

well with the input size. For that reason, several algorithms have been developed to tackle various input sizes and graph types, but many of them are based on Dijkstra's invention. Some approaches use extra information about the given graph such as a geographical or physical embedding, whereas other fairly recent ideas aim to delegate parts of the search for a shortest path into a preprocessing step, where the graph is prepared for the query to obtain better runtimes.

An approach requiring extra information about the graph was introduced by Hart, Nilsson and Raphael [HNR68], who developed a goal-directed technique called A* in 1968. This algorithm uses a potential function which underestimates the actual distance from each node to the target node to guide the search toward the latter. An obvious example for such a function on road networks is the great circle distance, as it certainly underestimates the actual travel distance. Unfortunately, in practice this yields a poor underestimator, as shown in [GH04]. One possible reason for this is that road networks exhibit a more grid-like topology, and hence subpaths of the shortest route might not follow the geodesic connecting the path's endpoints. Furthermore, natural obstacles such as mountain ranges or rivers may obstruct a path. As these are less of a problem in flight, A* is still an attractive choice; we will in the course of this thesis introduce A* and design a problem-specific potential function which yields a speedup over Dijkstra's algorithm.

In recent years, many speedup techniques have been developed which take but a fraction of the time used by Dijkstra's algorithm, at the expense of space- and time consuming preprocessing. One of them is based on A*: one chooses a (small) set of landmarks and precomputes the distances between them and all other nodes within the graph, by running all-to-one Dijkstra searches. Together with the triangle inequality, this yields a set of potential functions for the graph, all yielding lower bounds for the actual distance from any node to the target node. As the authors note in [BDG+15], one usually picks the maximum of these values if several potential functions are available, leading to the ALT algorithm (for A*, *L*andmarks and the *T*riangle inequality). The main intricacy in this approach is to find a good set of landmarks, which should lie close to a vast number of shortest paths. One can combine ALT with hierarchical approaches for speedup, see [BDS+08].

Contraction Hierarchies (CHs) as introduced in [GSSV12] as well as its time-dependent sibling, Time-Dependent Contraction Hierarchies (TCHs) (see [BDSV09]), is a very modern approach toward routing. Both variants focus on preprocessing the whole graph and introducing a number of shortcut edges, thus speeding up the query by a considerable factor. They are among the fastest algorithms for the shortest path problem, which renders them a natural choice to be considered. While they have been extensively studied on road networks, to the best of our knowledge, they have not yet been used in flight planning. Our experiments show that their speedup drops significantly when they are applied to this problem.

Hub Labeling is an approach which in a preprocessing step computes distances labels.

These distance labels are then scanned in the query stage in order to find a shortest path, thus processing but a fraction of the vertices of a graph. It was expanded to Hierarchical Hub Labeling by Abraham et al in [ADGW12]. While this approach yields excellent query times on such diverse graphs as road networks or small world graphs (compare [DGPW14]), to the best of our knowledge, there exists no time-dependent version of Hierarchical Hub Labeling, thus rendering the algorithm unsuitable for our problem.

Another approach combines hierarchies and special labels on the arcs, called arc flags, to lead the query toward the goal, thus giving rise to the SHARC algorithm. It can be extended to a time-dependent version, as is shown in [Del08]. However, in [DW09] the authors deem other hierarchical algorithms as TCHs superior to SHARC due to their lower preprocessing times.

Due to the nature of our problem, excessive preprocessing is prohibitive; in particular, computing distance tables together with a lookup query is not feasible, as weather conditions may render routes unattractive and computing times would be long anyway. For that reason, we will in this thesis concentrate on Dijkstra's algorithm, A* and Contraction Hierarchies, and compare their performances.

1.4 Our Contributions

We contribute to solving the Flight Trajectory Optimisation Problem by designing a new version of the A* algorithm. On road networks with static arc costs, A* does not lead to good results and is outperformed by other methods such as Contraction Hierarchies (CHs). On the Airway Network, however, our static version of A* yields a speedup competitive with that of CHs, but, as opposed to the latter, does not need any preprocessing.

A possible reason for these results is that although its design shares many facets of road networks, the Airway Network is essentially different in some aspects such as size – instances of the Airway Network have roughly 100 000 nodes and 350 000 arcs – average degree, or in that shortest paths have much fewer edges than in a road network.

In our setup, also the influence of wind on a route is very important: tailwinds push aircraft along their routes, headwinds and crosswinds slow them down. As winds change over time, this renders the problem highly time-dependent. Therefore, we also introduce two time-dependent solutions to the problem: one using the exact travel time functions on the arcs, and another one interpolating these through piecewise linear functions. Since time-dependency requires the FIFO property to be satisfied, we propose a criterion by which to check whether the resulting travel time functions satisfy this property.

As part of the design of the time-dependent A* algorithm, we introduce a feasible potential function underestimating the travel time from each node to the target. As computing the exact minimum travel time is rather costly, we underestimate it through Super-Optimal Wind, and artificial wind vector which is at least as good as all possible

wind vectors for an arc. This Super-Optimal Wind computation can be done very efficiently: we prove that it is very good in the sense that its absolute error with respect to the actual optimum is bounded linearly in an a-priori error estimate, and in practice, our computations show that it is in fact an excellent underestimation, with a relative error of less than 0.5%. Furthermore, the Super-Optimal Wind can be computed fast, taking just 5.6 seconds for all arcs.

Computationally, we show that in the Airway Network, contrary to road networks, the time-dependent version of A* dominates Time-dependent Contraction Hierarchies (TCHs) by more than one order of magnitude in terms of query time, while maintaining much lower preprocessing times. Using the potential function resulting from the Super-Optimal Wind, our version of the time-dependent A* algorithm solves a shortest path query in an average runtime of just 4.5 ms.

1.5 Outline

As to how this thesis is structured, we proceed as follows: in the next section, we lay out the notation for the rest of the thesis. In the subsequent Chapter 2, we shall introduce the problem in a more rigorous way. We will explain how we model it and which parts of the problem are of the most interest to us. Chapter 3 sees the theoretical introduction of Dijkstra's algorithm, A* and CHs. We motivate and describe them before proving their correctness.

In Chapter 4, we begin by introducing a criterion by which to check whether our choices of travel time functions satisfy the FIFO property. We then go on to design Super-Optimal Wind, a problem-specific underestimator to be used as a potential function by A*. Its quality is asserted both in theory and in practice. Our computational results are found in Chapter 5, where we assess the quality of A* and CHs and their time-dependent sibling on real world instances. We end this thesis with a few concluding remarks in Chapter 6.

1.6 The Notational Ground

In this section, we shall develop the notation we are going to use throughout this thesis. We assume that the reader is familiar with basic graph theory; a comprehensive introduction can for instance be found in [Die10]. Assume we are given a directed graph $G = (V, A)$. A **path** P will always be an ordered sequence of nodes

$$P = (u_0, u_1, \ldots, u_n), \quad u_i \in V,$$

or, in the time-dependent case, a sequence of pairs

$$Q = ((u_0, \tau_0), (u_1, \tau_1), \ldots, (u_n, \tau_n))$$

such that τ_i is the time at which u_i is reached. To find a shortest path, we also have to equip G with an arc weight function. In the following, a **static weight function** is a function

$$d\colon A \to [0, \infty)$$

mapping each arc to its non-negative length. Occasionally, we will interpret d as a function $V \times V \to [0, \infty)$, such as in the following: the **length $\ell(P)$ of a (static) path** $P = (u_0, u_1, \ldots, u_n)$ shall be given by

$$\ell(P) = \sum_{i=0}^{n-1} d(u_i, u_{i+1}).$$

Especially in Chapter 3, we will make use of the **distance function**

$$\mathrm{dist}\colon V \times V \to [0, \infty),$$

which takes two nodes s, t into the length of a shortest s-t-path $P = (s, u_1, \ldots, u_{n-1}, t)$. This yields

$$\mathrm{dist}(s, t) := \ell(P) = \ell((s, u_1, \ldots, u_{n-1}, t)).$$

Contrary to the above, in the time dependent case, the weight on any arc is given by a **travel time function**, or TTF, $T\colon A \times [0, \infty) \to [0, \infty)$ which maps an arc a and a starting time τ to the time $T(a, \tau)$ it takes to travel along that arc. We will occasionally switch between the version given above and the parametrised family of travel time functions

$$T_a\colon [0, \infty) \to [0, \infty).$$

In both cases (static or time-dependent), a pair (G, d) (or (G, T) in the time-dependent case) of graph and arc weight function is called a **cost graph**.

Assume now that $P = (u_0, u_1, \ldots, u_n)$ is a path in G, and we start to travel along P at time τ_0. The total time along P is then defined recursively via

$$T(P, \tau_0) := \sum_{i=1}^{n} T((u_{i-1}, u_i), \tau_{i-1}),$$

where

$$\tau_i = \tau_{i-1} + T((u_{i-1}, u_i), \tau_{i-1}) \text{ for } i = 1, \ldots, n.$$

When looking for a shortest path P between two nodes $s, t \in V$, we are looking for one which is shortest with respect to T, i.e., where $T(P, \tau_0)$ is minimised [1].

[1] this is equivalent to minimising the arrival time.

Given a cost graph (G, d) with the static cost function $d\colon A \to \mathbb{R}_0^+$ and two nodes $s, t \in V$, the problem of finding a minimum-cost path between s and t is called a **Shortest Path Problem**, or SPP for short. If we exchange d for its time-dependent counterpart $T\colon A \times \mathbb{R}_0^+ \to \mathbb{R}_0^+$, the same problem is called a **Time-Dependent Shortest Path Problem**, which we will denote by TSPP.

Last but not least, a **query** shall denote the actual run of any search algorithm to determine a shortest path between two given vertices s, t. Any additional data the query needs in order to obtain a correct result will in some cases be achieved through a **preprocessing** step (compare also Chapter 3). While there exist algorithms such as CHs that do not require any knowledge of s or t for preprocessing, e.g. computing a potential function for A* in our setting is a preprocessing step where knowledge of the target node is needed.

2 The Horizontal Flight Trajectory Optimisation Problem

In this chapter, we shall lay the foundation of this thesis by introducing the Horizontal Flight Trajectory Optimisation Problem (HFTOP). We begin by explaining the more general Flight Trajectory Optimisation Problem (FTOP), before restricting ourselves to the case of HFTOP. We will also discuss modelling the problem as a Shortest Path Problem.

To clarify the terms we are about to use, we shall begin this chapter with a recollection of some fundamental aeronautical notions.

2.1 An Aeronautics Primer

In this section, we recall the basic concepts connected with aeronautics as far as they shall be used here. Several notions will reappear throughout the thesis, which is why we shall introduce them here; for a much more thorough introduction, see [HSS14].

The length between any two points on the earth is the **ground distance**, typically denoted by d_G. It is a constant and unaffected by wind. The **airspeed** v_A of an aircraft is defined as its speed relative to the air mass surrounding it. In general, pilots can change the airspeed of their aircraft, as reflected in FTOP. In HFTOP, however, we shall assume that v_A is constant. Connected with the airspeed is the **air distance** d_A, which is the distance an aircraft travels relative to the surrounding air mass. Unlike the ground distance, it is not constant if we assume changing wind conditions. The same holds for the **ground speed** v_G, which constitutes an aircraft's speed with respect to the ground. It is also highly volatile and dependent on wind conditions (see Section 2.3.2).

2.2 The Flight Trajectory Optimisation Problem

The **Flight Trajectory Optimisation Problem** (FTOP) seeks to find a minimum cost trajectory between any two airports s, t in the Airway Network. The Airway Network consists of a set of predefined coordinates called **waypoints**, some having navigational aids, some being arbitrarily defined (compare [KASS12]). Waypoints are connected by **segments**, straight lines between them. The projection of an aircraft's trajectory to the surface of the earth must adhere to segments and waypoints just like a car must adhere

to the roads, and cruising is only allowed on a predefined set of altitudes, so-called **flight levels**. These go up to 41 000ft (approximately 12 500m), with most flights flying higher than 25 000ft (≈ 7 600m) and only very short flights flying lower in some circumstances. These multiple levels or layers of the Airway Network render it rather difficult to visualise – however, a good projection is shown on skyvector.com and in Figure 2.1.

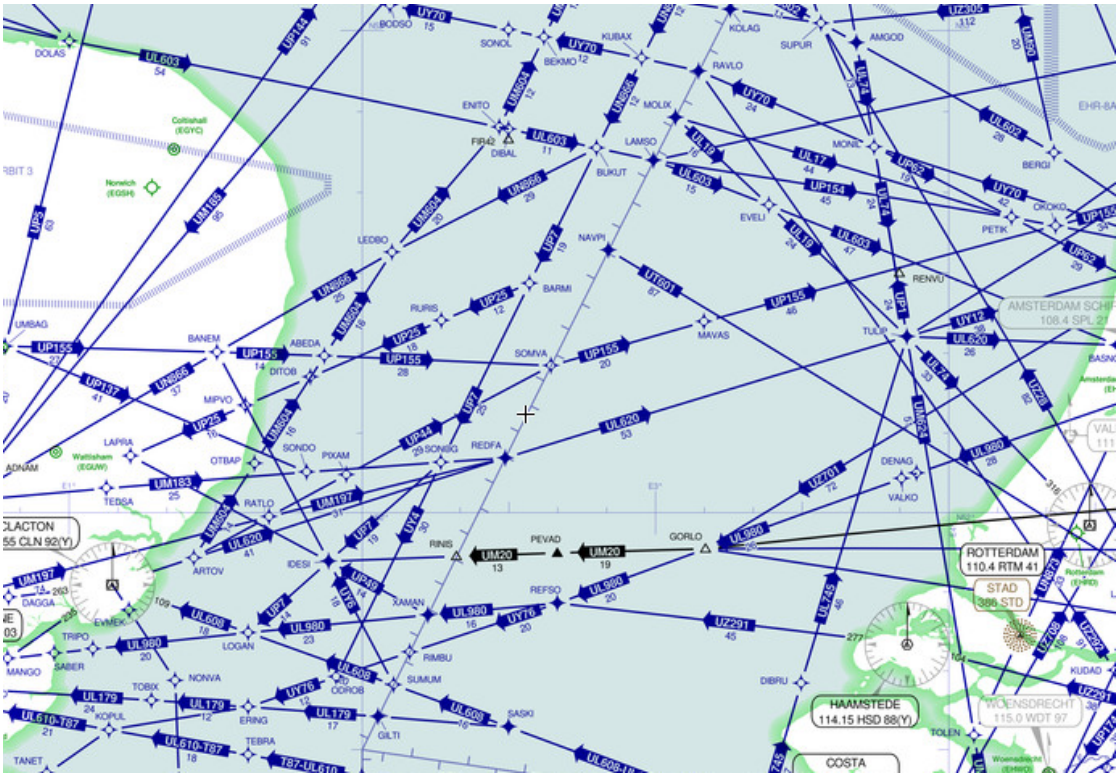


Figure 2.1: Airway Network over the North Sea, as taken from skyvector.com

To evaluate the costs of a trajectory, one introduces a cost function consisting of several parts: the most obvious one is the time an aircraft is airborne. Depending on the route, an aircraft spends more or less time in the air, and one usually aims to minimise this **estimated time enroute** (ETE). The **fuel cost** constitutes another part of the cost function: the heavier an aircraft, the more fuel it burns, and depending on whether an aircraft climbs, descends or is in level flight, its fuel consumption varies. One tries to load as little fuel as possible, as every kilogramme of superfluous fuel unnecessarily impacts the environment, and decreases efficiency by rendering the aircraft heavier.

The fuel efficiency of an aircraft is restricted by its performance limits, in particular, aircraft fly more efficiently at high altitudes. This impacts the vertical profile of a route, as one often wishes to climb whenever the weight of an aircraft allows it.

One aspect of FTOP which is maybe less obvious, albeit important, is the problem of overfly costs. The Airway Network is divided into regions called **airspaces** which are typically assigned to a country or a group of those. Each time an aircraft flies through an airspace, it incurs **overfly charges**, which add to the overall cost. An accurate cost function should incorporate all three aspects, as a weighted sum. A shortest travel time path need not yield the minimum overfly costs (and rarely will) in the same way as an optimal overfly charges path need to be minimal with respect to ETE or fuel costs.

Furthermore, there exist several restrictions when it comes to flight planning. These range from keeping clear of politically unstable or high-risk areas to avoiding overcrowding in airspaces, thus reducing workload on Air Traffic Control. These latter restrictions are for example published by EUROCONTROL in the Route Availability Document [Eur16], and are frequently updated. The difficulty in tackling them is increased further as some restrictions only apply to the choice of source and target airports s and t, depend on an airspace or are only valid for a certain time period.

A flight is made up out of five phases, namely TAKE-OFF, CLIMB, CRUISE, DESCENT and APPROACH. The arguably most important one is the CRUISE phase, in which the aircraft flies levelly. When necessary to change altitude, one can initiate either a CLIMB or a DESCENT phase, which take the aircraft to any higher or lower altitude respectively. Usually, these phases directly follow (or in the case of DESCENT, precede) the TAKE-OFF and APPROACH phases, but can in principle occur anytime in-flight. Since aircraft fly more efficiently at high altitudes, a vertical profile of a flight may include several climbs whenever an aircraft becomes light enough to fly higher. On the other hand, it is sometimes necessary to descend to a lower altitude to avoid inclement weather or overcrowding of airspaces, thus creating multiple DESCENT phases in a flight. The Airway Network reflects these five phases by consisting of multiple layers of the same waypoint set; on each layer, almost the same set of segments connects the waypoints. There are, however, some segments which are only allowed on certain altitudes[1]. These phases add the extra constraints to FTOP that a trajectory must begin at the source airport's altitude and end at the target airport's altitude.

2.3 The Horizontal Flight Trajectory Optimisation Problem

As tackling FTOP in its full complexity is beyond the scope of this thesis, we consider a simplified version of FTOP. Firstly, we do not heed any constraints leading to possible path constrictions. Secondly, we disregard additional costs such as overfly costs – for a treatise on these, we refer the reader to [MdlC16]. We add the assumption to FTOP that we are only allowed to fly on one altitude (or layer), thus disregarding CLIMB or DESCENT phases. Indeed, we assume that every flight starts at the flight altitude and ends there.

[1]For example, segments over mountain ranges must have a minimum altitude to clear them.

We further assume that the airspeed is constant, i.e., neither vertical optimisation nor a speed profile is considered.

For two reasons, this is a good choice. The first one stems from the 2D+2D approach by Karisch et al discussed above: in order to obtain a valid route, one first optimises horizontally. Hence, in Karisch's approach, our simplification is an integral part of the solution to the full problem.

A second reason for considering only one layer is that especially for long-haul flights, the CRUISE phase dominates the others in terms of time. It is therefore the most important one and also one where optimising can yield potential savings for the airline.

We consider a path P between s and t **optimal** when the time an aircraft spends airborne on P is minimal, i.e., the cost function consists only of the ETE. Although this may seem to disregard fuel costs, they are in fact part of the optimisation: since we assume that we fly with constant airspeed and neither climb nor descend, the fuel consumption is directly equivalent to the travel time.

Since we impose that only horizontal flight is allowed, we call our simplification of FTOP the **Horizontal Flight Trajectory Optimisation Problem** (HFTOP).

2.3.1 Modelling HFTOP

When mathematically modelling HFTOP, we identify the Airway Network with a directed graph $G = (V, A)$, where we interpret the waypoints as nodes $v \in V$ and the segments as (directed) arcs $a \in A$. In the following, we will interchangeably write both waypoints for nodes and segments for arcs.

We assume that we are given weather prognoses for some times $\mathcal{T} = \{t_0 < \ldots < t_r\}$, such that every flight's duration is contained in $[t_0, t_r]$. To model the time dependency in HFTOP, we consider three variants. In the first one, we assume that no wind is blowing, hence the distance function on the arcs will be the ground distance (cf Section 2.3.2). This **static case** is exactly a (static) Shortest Path Problem.

In the second and third approach, we model any wind dependency as a function of the time (see again Section 2.3.2). In this fashion, the length of a segment varies through time as wind conditions change, which allows us to model HFTOP as a Time-Dependent Shortest Path Problem. There are, however, two different ways to model the travel time. In the first one, which we shall denote the **exact case**, we compute the travel time on the arcs exactly.

While this version already turns out to be very fast, we seek to improve the runtime by approximating the travel time function through a piecewise linear function, or PWL for short (for a thorough definition, see Section 2.3.2), yielding what we shall call the **PWL case**. As will become apparent in Chapter 5, they are indeed faster; however, as weather forecasts do not allow for a natural way to represent a travel time function as a PWL, we include an error discussion in the same chapter. Since the travelled time is directly proportional to the fuel consumption, computational exactness is paramount

for safety, ultimately rendering the PWL case less favourable than the exact case. As we will see in Chapter 5, also the computational speedup of the PWL case over the exact case does not justify the time to create the piecewise linear functions out of our exact data in the first place.

2.3.2 How To: Obtain a TTF

Since we are considering several different problems, we have a variety of different travel times, which we will introduce in this section.

The Static Case

Since we assume that the airspeed is constant and that in the static case, there is no wind, the travel time along an arc $a = (u, v) \in A$ does not depend on when we enter it. Instead, the airspeed v_A is linked to the air distance d_A via the formula $d_A = v_A \cdot t$, so minimising the travel time is equivalent to minimising the air distance. The air distance is in turn exactly the ground distance of a, for we assume that no wind is blowing. This is why in the static case, the travel time function will be the ground distance.

The Time-Dependent Formula

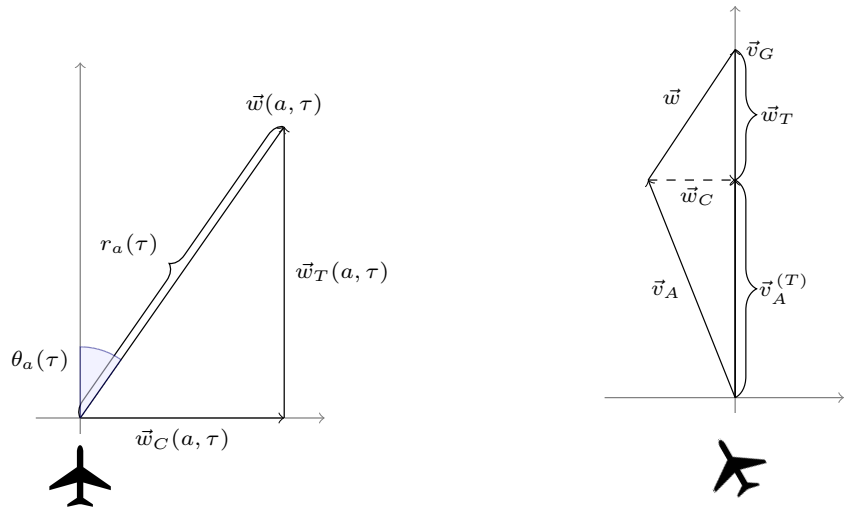
As becomes apparent from Figure 1.1, an optimal path is very much subject to the prevailing wind conditions, which we model as a function of the time. To this end, we fix an arc $a \in A$ and a time τ and represent the wind as a vector $\vec{w} = \vec{w}(a, \tau)$ with a **crosswind component** $\vec{w}_C(a, \tau)$ and a **trackwind component** $\vec{w}_T(a, \tau)$, which are the components of $\vec{w}$ with angles $\pi/2$ and 0 relative to a's direction, respectively (cf Figure 2.2a). Note that, as is common in aviation, all angles are measured clockwise and with respect to the heading of the aircraft.

We will for the course of the whole thesis assume that wind vectors are given as polar vectors, so let $\vec{w}$ have the representation $\vec{w}(a, \tau) = (r_a(\tau), \theta_a(\tau))$, where $r_a(\cdot)$ is the wind force on a and $\theta_a(\cdot)$ is the angle of $\vec{w}$ relative to a's direction. This means that the magnitude of the components of $\vec{w}$ can be expressed as

$$w_C(a, \tau) := \|\vec{w}_C(a, \tau)\| = r_a(\tau) \cdot \sin \theta_a(\tau) \text{ and}$$
$$w_T(a, \tau) := \|\vec{w}_T(a, \tau)\| = r_a(\tau) \cdot \cos \theta_a(\tau)$$

Now suppose that as in Figure 2.2b, an aircraft is travelling along the arc a at time τ with constant airspeed $\vec{v}_A$ and is subject to a wind vector $\vec{w}(a, \tau)$. To avoid being carried off to the side, an aircraft alters its angle of attack such that the resulting ground speed vector

$$\vec{v}_G(a, \tau) = \vec{v}_A(a, \tau) + \vec{w}(a, \tau)$$



(a) A wind vector $\vec{w}$ and its components $\vec{w}_C$ and $\vec{w}_T$

(b) An aircraft flying at speed $\vec{v}_A$, influenced by the wind vector $\vec{w}(a,\tau)$

Figure 2.2: Wind and Ground Speed Terminology

points in the direction of the true track of a. Through this alteration, the above equation can be written as

$$\vec{v}_G(a,\tau) = \vec{v}_A(a,\tau) + \vec{w}(a,\tau) = \vec{v}_A^{(T)} + \vec{w}_T(a,\tau),$$

where $\vec{v}_A^{(T)}$ denotes the track component of $\vec{v}_A$, i.e. the component of $\vec{v}_A$ with angle 0 relative to a's direction (see also Figure 2.2b). We then find

$$v_G(a,\tau) := \|\vec{v}_G(a,\tau)\| = \|\vec{v}_A^{(T)}(a,\tau)\| + \|\vec{w}_T(a,\tau)\| = \sqrt{v_A^2 - w_C^2(a,\tau)} + w_T(a,\tau).$$

The travel time $T(a,\tau)$ along a is now the solution of the equation

$$d_G^a = \int_\tau^{\tau+T(a,\tau)} v_G(a,\sigma)\,\mathrm{d}\sigma.$$

Since this integral is computationally very expensive to solve, we will instead assume that the wind conditions on the segment depend only on the time τ at which the aircraft is entering it. This is equivalent to writing

$$v_G(a,\sigma) \equiv v_G(a,\tau) \text{ for all } \sigma \in [\tau, \tau + T(a,\tau)],$$

which reduces the formula to

$$T(a,\tau) = \frac{d_G^a}{v_G(a,\tau)} = \frac{d_G^a}{\sqrt{v_A^2 - w_C^2(a,\tau)} + w_T(a,\tau)}. \tag{2.1}$$

In practice, we have to make some further ammendments: in our application, we are only given the weather prognoses for a set $\mathcal{T} = \{t_0, t_1, \ldots, t_r\}$ of prognosis times, all spaced three hours apart. So, we have to compute the values $w_C(a, \tau)$ and $w_T(a, \tau)$ for any $\tau \notin \mathcal{T}$. We do so by polarly interpolating the nearest neighbours of τ in $\mathcal{T}$: say $\tau \in (t_i, t_{i+1})$ for some $i \in \{0, \ldots, r-1\}$. For simplification, we set $r_i^a := r_a(t_i)$ and $\theta_i^a := \theta_a(t_i)$, as well as

$$\lambda(\tau) = \frac{t_{i+1} - \tau}{t_{i+1} - t_i}.$$

The cross- and trackwind components are then

$$w_C(a, \lambda(\tau)) = (r_{i+1}^a + \lambda(\tau)(r_i^a - r_{i+1}^a)) \cdot \sin(\theta_{i+1}^a + \lambda(\tau)(\theta_i^a - \theta_{i+1}^a)) \tag{2.2}$$
$$w_T(a, \lambda(\tau)) = (r_{i+1}^a + \lambda(\tau)(r_i^a - r_{i+1}^a)) \cdot \cos(\theta_{i+1}^a + \lambda(\tau)(\theta_i^a - \theta_{i+1}^a)). \tag{2.3}$$

It makes sense to assume that wind always turns via the smaller of the two possible angles. In particular, in Figure 2.3, we expect that the wind turns via the blue angle rather than the red one. For our interpolation, we assume the same: suppose we are given θ_1, θ_2, and wlog $\theta_2 > \theta_1$, as suggested in the figure (recall that angles are measured clockwise and from the top axis). If $\theta_2 - \theta_1 > \pi$, then instead of θ_2, we consider $\tilde{\theta}_2 = \theta_2 - 2\pi$. Thus, we can make sure that $|\theta_2 - \theta_1| \leq \pi$ always holds.

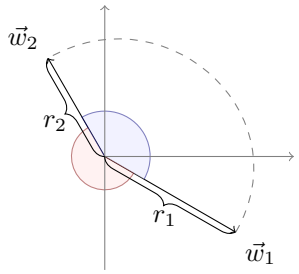


Figure 2.3: Interpolation via the smaller angle (blue)

All of the above simplifications are state-of-the-art in the industry. Inserting (2.2) and (2.3) into (2.1), one obtains the function as depicted in blue in Figure 2.4, where the travel time is shown for some arc $a \in A$ over an period of 15 hours.

The PWL Approach

Assume we are given a discretisation $t_0 = \tau_0 < \tau_1 < \ldots < \tau_N = t_r$ of $[t_0, t_r]$, and define $I_i = [\tau_i, \tau_{i+1}]$. Then a **piecewise linear function** (PWL) is a continuous function

$$f \colon [t_0, t_r] \to \mathbb{R}$$

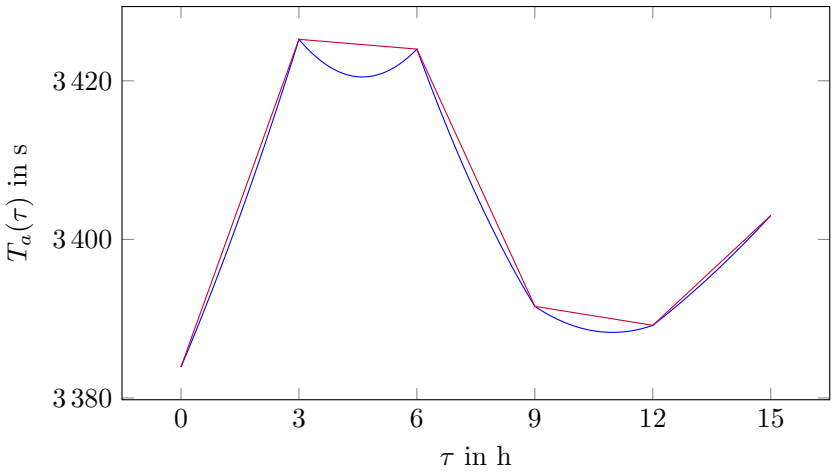


Figure 2.4: The exact travel time function (blue) and a PWL approximation (purple)

such that the restriction

$$f_i = f|_{I_i} \colon [\tau_i, \tau_{i+1}] \to \mathbb{R}$$

is linear. In our setup, we further require that $\tau_i - \tau_{i-1} = \Delta$ for all $i = 1, \dots, N$, and that $r|N$. The latter is necessary since we are given the forecast times $\{t_0, t_1, \dots, t_r\}$, and want any interval I_i to be fully contained in one $[t_j, t_{j+1}]$ for some $j \in \{0, \dots, r-1\}$.

For each τ_i, we can now precompute the travel times $T_a(\tau_i)$ for each i on all segments a, which are then the exact travel times. In case we want to calculate a travel time $\tau \in (\tau_i, \tau_{i+1})$, we linearly interpolate $T_a(\tau_i)$ and $T_a(\tau_{i+1})$ to obtain an approximation of the travel time. To be more precise, let $\lambda \in [0, 1]$ be such that $\tau = \lambda\tau_i + (1-\lambda)\tau_{i+1}$. We then define

$$\tilde{T}_a(\tau) := \lambda T_a(\tau_i) + (1-\lambda) T_a(\tau_{i+1}). \tag{2.4}$$

This is a piecewise linear function, and promises to be much faster computation-wise. Note that although in both cases we interpolate linearly, this approach is essentially different from the exact one. In the exact approach, we only interpolate the wind vectors – here, we additionally linearise the travel times between the τ_i. The differences in the approaches are depicted in Figure 2.4.

3 Algorithms for the Shortest Path Problem

In the last chapter, we have seen that HFTOP can be modelled as a (time dependent) shortest path problem. For this reason, we shall in this part of the thesis describe algorithms to solve SPP and TSPP from a theoretical point of view and justify their correctness.

3.1 Dijkstra's Algorithm

Conceived by E. Dijsktra in[Dij59], it is one of the oldest and most well-known shortest path algorithms. As it is the main ingredient for both of our speedup techniques and serves as a reference algorithm to compare our results to, we shall introduce it here and prove its optimality.

3.1.1 The Static Case

Let $G = (V, A)$ be a graph, and $s, t \in V$ two distinguished nodes in G. Further, let $d\colon A \to [0, \infty)$ be a weight function on the arcs; notice that by our definition, arc weights can never be negative. We are looking for a shortest path from s to t with respect to d. To this end, we introduce the function

$$d_s\colon V \to [0, \infty]$$

which assigns each node a tentative distance from s. At the beginning of the search, each node $v \in V \setminus \{s\}$ is assigned the distance ∞, which is decreased in the course of the algorithm. To ease notation, we define the **out-neighbours** of $u \in V$ as

$$N_G^+(u) = \{v \in G\colon \exists (u, v) \in A\}.$$

We will call a node u **processed** if its outgoing arcs have been examined and all $v \in N_G^+(u)$ are assigned tentative distances. A node v having a tentative distance $d_s(v) < \infty$ which is not yet processed will be called **reached**. Thus, we can keep two disjoint sets: firstly, the set $\mathcal{Q} \subset V \times [0, \infty]$ of reached, but not yet processed nodes together with their distances, and secondly $\mathcal{P}$ as the set of processed nodes. At the beginning, we let $\mathcal{Q} = \{(s, 0)\}$ and $\mathcal{P} = \emptyset$. Note that we can always weakly order $\mathcal{Q}$ by the distances, thus obtaining a priority queue of reached nodes. In each step, the node u with the least distance is removed from $\mathcal{Q}$ and inserted into $\mathcal{P}$. Then every neighbour v of u is examined, where several cases may occur:

(i) $v \notin \mathcal{P}, \mathcal{Q}$. If that is the case, then v has not yet been reached. We insert it into $\mathcal{Q}$ with the distance $d_s(u) + d(u, v)$.

(ii) $v \in \mathcal{Q}$. Here, we have to check whether $d_s(v) > d_s(u) + d(u, v)$ holds. If this turns out to be true, the path reaching v from s via u is shorter than any other path found so far – we update $d_s(v)$ to its new value $d_s(u) + d(u, v)$.

(iii) $v \in \mathcal{P}$, so v is already processed. In that case, we disregard the arc (u, v) and go on to the next neighbour.

If u is t, we stop the algorithm; if one keeps additional data such as predecessor nodes, then one is also able to retrieve a shortest s-t-path. In pseudocode, Dijkstra's algorithm can be written down as follows:

Algorithm 1: Dijkstra's Algorithm

Input : Graph $G = (V, A)$,
Cost function $d\colon A \to \mathbb{R}_0^+$
Output: Shortest path distance $d_s(t)$
Data : Priority queue $\mathcal{Q}$, set $\mathcal{P}$ of processed nodes

```
1  Q = {(s, 0)};
2  d_s(v) = ∞ for all v ∈ V \ {s};
3  while Q ≠ ∅ and t ∉ P do
4      (u, d_s(u)) = Q.removeMin();
5      for v ∈ N_G^+(u) do
6          if v ∈ P then go to line 5 ;
7          altDist = d_s(u) + d(u, v);          // alternative distance via u
8          if v ∈ Q and d_s(v) > altDist then
9              Q.update(v,altDist);
10         else if v ∉ Q then
11             Q.insert(v,altDist);
12     P.insert(u);
13 return d_s(t);
```

The proof that Dijkstra's algorithm is correct is standard and can for instance be found in [KN12]. We present our own version in the following theorem.

Theorem 1. *Dijkstra's Algorithm yields a shortest path between s and t if it exists.*

Proof. We claim that for any processed node u, $d_s(u)$ is the distance of the shortest path from s to u, i.e., if $u \in \mathcal{P}$, then $d_s(u) = \text{dist}(s, u)$. To prove it, we use induction on the number of processed nodes: it is clear that $d_s(s) = 0$ is indeed the distance from s to s. So, assuming the statement holds for $|\mathcal{P}| = n$ processed nodes, we have to show that it holds for $n + 1$ processed nodes as well. Note that $|\mathcal{P}|$ increases in every step.

Let $\mathcal{P}$ denote the set of processed nodes such that $|\mathcal{P}| = n$, and say u is the node about to be added to $\mathcal{P}$, so define $\mathcal{P}' = \mathcal{P} \cup \{u\}$. Let P be a shortest path from s to u, and suppose

$$\ell(P) = \text{dist}(s, u) < d_s(u).$$

Since P starts in s, it starts in the set $\mathcal{P}$. At some point, however, it must leave $\mathcal{P}$, since $u \notin \mathcal{P}$. Let (v, w) be the first edge to traverse from $\mathcal{P}$ to $V \setminus \mathcal{P}$, i.e. $v \in \mathcal{P}$ and $w \notin \mathcal{P}$. We let $P_v = (s, \dots, v)$ be the subpath of P which is entirely contained in $\mathcal{P}$; then clearly

$$\ell(P) \geq \ell(P_v) + d(v, w) = \text{dist}(s, v) + d(v, w) \geq d_s(v) + d(v, w),$$

where the last inequality holds because of the induction hypothesis and since $v \in \mathcal{P}$. As w is incident to the processed node v, it must already have been updated, and $d_s(w) \leq d_s(v) + d(v, w)$. Since u is the node about to be added to $\mathcal{P}$, we find $d_s(u) \leq d_s(w)$. Combining these two inequalities with the ones above yields the contradiction

$$d_s(u) > \text{dist}(s, u) = \ell(P) \geq d_s(v) + d(v, w) \geq d_s(w) \geq d_s(u).$$

Hence, the claim must be true and $d_s(u) = \text{dist}(s, u)$. This also means that once t is processed, we can stop the search. □

Note that we stipulated $d\colon A \to [0, \infty)$. While in our application, this always holds true, the following very simple example shows what could happen if we allowed negative arc costs.

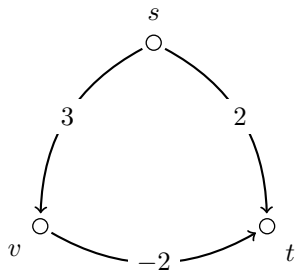


Figure 3.1: Dijkstra's algorithm fails for negative arc costs

Example 2. Assume we want to find a shortest path from s to t in the graph in Figure 3.1. If we apply Dijkstra's algorithm, we find the shortest path (s, t) with length 2, as node t is processed right after node s. But clearly, the route (s, v, t) has length 1 and is thus shorter.

We will also consider several variations of Dijkstra's Algorithm: first, by omitting the condition $t \notin \mathcal{P}$ in line 3 of Algorithm 1, we force it to run until $\mathcal{Q}$ is empty. This happens if and only if every (reachable) node in the path component of s has been processed. Consequently, we call this version of Dijkstra's algorithm **one-to-all**. For another variant, replace $N_G^+(u)$ by $N_G^-(u)$ in line 5, where

$$N_G^-(u) = \{v \in G \colon \exists (v, u) \in A\}$$

is the set of **in-neighbours** of u. If one now starts the query with $\mathcal{Q} = \{(t, 0)\}$ and uses $d_t(\cdot)$ instead of $d_s(\cdot)$, then one obtains what we call an **all-to-one** Dijkstra – we simply reverse the search direction from the source node to the target node. This variant will be used in A* preprocessing and is therefore of great interest to us.

3.1.2 Bidirectional Dijsktra

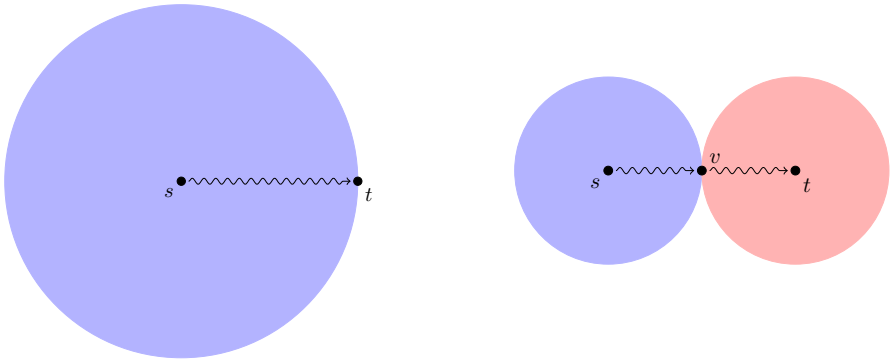
When running an s-t-Dijkstra, every node v with $0 \leq \text{dist}(s, v) \leq \text{dist}(s, t)$ will be examined. For any query algorithm $\mathcal{A}$, we define the **search space** $\mathcal{S}_\mathcal{A}$ of $\mathcal{A}$ to be the set $\mathcal{S}_\mathcal{A} = \mathcal{P}$ after $\mathcal{A}$ terminates. Especially on long routes, $\mathcal{S}_\mathcal{A}$ can be quite extensive, often encompassing nodes which are not 'in the right direction'. One way of confining the search space is by running two (quasi-) simultaneous Dijkstra searches, one from s and one from t. Whenever their respective scopes meet, we can set up a tentative path, and once a node is processed in both scopes, we can end the search. The advantage is that we have to visit only roughly half of the nodes, since instead of one search disc with radius $\text{dist}(s, t)$, we obtain two discs, each with radius $\frac{\text{dist}(s,t)}{2}$ (cf Figure 3.2).

The query of Contraction Hierarchies (Section 3.3) is based on a bidirectional search, which is why we introduce it here.

3.1.3 The Time-Dependent Case

The time-dependent case was first tackled by Cooke and Halsey in [CH66], in which they give a dynamic programming approach to the problem: they set up a functional equation which they then solve iteratively. Dreyfus noted in [Dre69] that the time-dependent shortest path problem can be solved by a natural extension of Dijktra's algorithm, and is thus very similar to the static case.

The generalisation to the time-dependent case works as follows: in Algorithm 1, instead of the tentative distance $d_s(u)$ one now keeps track of the tentative time $\tau_{s,\tau_0}(u)$



(a) Search space for unidirectional search **(b)** Search space for bidirectional search

Figure 3.2: Search spaces for two types of Dijkstra's algorithm

it takes to reach u when departing node s at time τ_0. Consequently, the alternative distance via u in line 7 now becomes an alternative time, namely

$$\texttt{altTime} = \tau_{s,\tau_0}(u) + T((u,v), \tau_{s,\tau_0}(u)),$$

and the subsequent lines are changed accordingly. The return value is $\tau_{s,\tau_0}(t)$, the time it takes to reach t from s when departing s at time τ_0. It was pointed out by Kaufmann and Smith in [KS93] that Dreyfus' extension only yields an optimal result if a kind of consistency assumption is not violated. We introduce this assumption as

The FIFO Property

The **first-in-first-out** or FIFO property roughly states that, in our case, aircraft cannot overtake each other on the same segment. Consider an arc a with the travel time function $T_a(\cdot) = T(a, \cdot)\colon [0,\infty) \to [0,\infty)$. Now suppose aircraft A enters the arc a at the time τ_A and spends $T_a(\tau_A)$ on the segment, and aircraft B enters the same segment at a later time $\tau_B > \tau_A$ and spends $T_a(\tau_B)$ on it, then we say T_a **satisfies the** FIFO **property** if we have

$$\tau_A + T_a(\tau_A) \leq \tau_B + T_a(\tau_B) \quad \text{whenever } \tau_A < \tau_B. \tag{3.1}$$

If the FIFO property is violated, one can no longer guarantee path optimality, as is shown in the next example.

Example 3. [KS93] Consider the graph as shown in Figure 3.3 with the assigned travel time functions, and say we want to find a shortest path from s to t, starting at time $\tau = 0$.

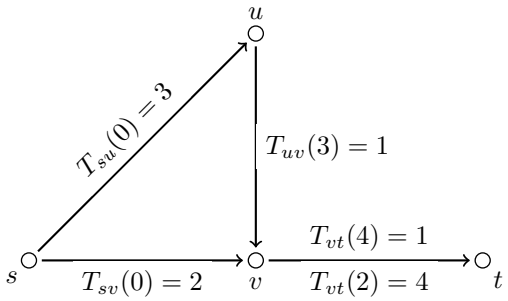


Figure 3.3: Example of a non-FIFO network [KS93]

If one were to use a straightforward generalisation of Dijkstra's algorithm in this case, one would obtain the path (s, v, t), and arrive at t at the time $\tau_t = 6$. But clearly, the path (s, u, v, t) is shorter, arriving at the time $5 < 6$. The key in this example is that T_{vt} does not satisfy the FIFO property, as we find

$$\tau_A + T_{vt}(\tau_A) = 2 + 4 > 4 + 1 = \tau_B + T_{vt}(\tau_B).$$

If the FIFO principle is violated but waiting at nodes is still allowed, the problem stays polynomially solvable, as shown in [OR90]. However, in our application it is for obvious reasons not possible to wait at waypoints, so if the Airway Network happened to be non-FIFO, we could not guarantee optimality of the paths. We propose criterions in Theorems 16 and 24 which guarantee that our choices of T_a are FIFO. Fortunately, our calculations show that these are satisfied in practice.

We have seen in Example 3 that although we can generalise Dijkstra's algorithm, it might not yield optimal results if the FIFO principle is violated. If, on the other hand, the FIFO property holds, then we can guarantee that the straightforward generalisation of Dijkstra's algorithm yields an optimal path:

Lemma 4. *[KS93, Lemma 1] Suppose the* FIFO *property holds, and let u_0, u_n be two vertices in (G, T). Then there exists a path $P = ((u_0, \tau_0), \ldots, (u_{n-1}, \tau_{n-1}), (u_n, \tau_n))$ such that*

(i) the time τ_n at which u_n is reached is minimal, and

(ii) the 1-truncation $((u_0, \tau_0), \ldots, (u_{n-1}, \tau_{n-1}))$ of P is a path such that the time τ_{n-1} at which u_{n-1} is reached is minimal.

Proof. Let P' be any shortest path from $(u_0, \sigma_0 = \tau_0)$ to (u_n, σ_n) via (u_{n-1}, σ_{n-1}), i.e.,

$$P' = ((u_0, \sigma_0 = \tau_0), \ldots, (u_{n-1}, \sigma_{n-1}), (u_n, \sigma_n)).$$

Suppose P_{n-1} a shortest path from (u_0, τ_0) to (u_{n-1}, τ_{n-1}) (note that P' and P_{n-1} need not share any inner vertices). We claim that the extension of P_{n-1} by u_n is a shortest

path from u_0 to u_n and that $\tau_n = \tau_{n-1} + T((u_{n-1}, u_n), \tau_n) = \sigma_n$. Note that since P_{n-1} is assumed to be optimal, $\tau_{n-1} \leq \sigma_{n-1}$ holds. Using the FIFO property, we find that

$$\tau_n = \tau_{n-1} + T((u_{n-1}, u_n), \tau_{n-1}) \leq \sigma_{n-1} + T((u_{n-1}, u_n), \sigma_{n-1}) = \sigma_n,$$

so $\tau_n \leq \sigma_n$. Since P' was assumed to be optimal, σ_n is minimal, and so τ_n must be minimal. This means that the path P obtained by following P_{n-1} and extending to u_n is the desired path. □

By recursively applying Lemma 4, we obtain

Corollary 5. *[KS93, Theorem 2] Suppose the* FIFO *property holds, and let u_0 and u_n be two vertices in (G, T). Then there exists a path $P = ((u_0, \tau_0), \ldots, (u_n, \tau_n))$ such that for every $i \in \{0, \ldots, n\}$, the time τ_i at which u_i is reached is minimal.*

Note that we have to stipulate that T actually maps every arc onto a non-negative travel time, thus disallowing negative cycles and guaranteeing that every path is finite – within our application, this is always the case.

We have just seen that there exist paths from u_0 to u_n such that every node u_i is reached at a minimum time τ_i, which is the foundation of Dijkstra's algorithm. In [KS93], the authors proceed to show that under the assumption of the FIFO property, any algorithm which solves the static problem optimally also solves the time-dependent version optimally. We state their result as

Theorem 6. *[KS93, Theorem 4] If the* FIFO *property holds, the straightforward generalisation of Dijkstra's algorithm to the time-dependent case yields an optimal path between any two nodes s, t on (G, T).*

The above theorem is fundamentally important for TSPP, as it guarantees that the problem is (under mild assumptions) polynomially solvable. In particular, it guarantees that algorithms which are derived from Dijkstra's algorithm yield optimal solutions in the time dependent case.

3.2 A*

The algorithm A* was introduced by Hart et al in [HNR68]. It seeks to speed up the query for a shortest path by altering the keys in the priority queue $\mathcal{Q}$ according to a potential function. To this end, it uses 'outside' information about the graph (e.g. a geographical embedding) or data obtained in a preprocessing step.

As an example, consider a flight from London to New York. We know the cities' geographical location, and that the shortest path connecting them will more or less follow the geodesic as close as possible within the Airway Network. Therefore, it makes little sense to examine waypoints over Greece, Finland or Austria, all of which are regions

contained in the search space of Dijkstra's algorithm. Instead, we can help the query algorithm by rendering nodes in the right direction 'more attractive'. This is usually done by defining a **potential function** $\pi\colon V \to \mathbb{R}_0^+$ such that π never overestimates the actual distance to the target node. This potential is then used to modify the keys in the priority queue. This section and theoretical results are based on [GH04] and [HNR68].

Definition 7. Let $G = (V, A)$ be a graph and $t \in V$ one of its nodes. Assume we are given a cost function d on G, yielding the cost graph (G, d). A function $\pi_t\colon V \to \mathbb{R}_0^+$ on the nodes is called a **potential** on (G, d) with respect to t. We further call π_t **admissible** whenever

$$\pi_t(v) \leq \operatorname{dist}(v, t) \quad \forall v \in V,$$

i.e., π_t never overestimates the actual shortest path length in G. If

$$\pi_t(v) \leq \pi_t(w) + d(v, w) \quad \forall (v, w) \in A$$

holds, then we call π_t **feasible**.

The second condition is a form of the triangle inequality, while the first condition implies that any admissible potential satisfies $\pi_t(t) = 0$.

Lemma 8. *Let $G = (V, A)$ be connected and π_t a potential with respect to some node t on (G, d) for some cost function d. Then if π_t is feasible and $\pi_t(t) = 0$, it is also admissible.*

Proof. Let $v \in V$ be any node and $(v, u_1, \ldots, u_n, t)$ a shortest path to t. Now clearly $\pi_t(v) \leq \pi_t(u_1) + d(v, u_1)$, and iterating this inequality yields

$$\pi_t(v) \leq \pi_t(t) + d(v, u_1) + \sum_{i=2}^{n} d(u_{i-1}, u_i) + d(u_n, t) = \operatorname{dist}(v, t) + \pi_t(t),$$

which proves the claim since $\pi_t(t) = 0$. □

Now suppose we are given a cost graph (G, d) and two nodes s, t in G. If we want to find a shortest path from s to t, then for any potential π_t, we can design the following algorithm: let

$$\begin{aligned} f\colon V &\to \mathbb{R} \\ v &\mapsto d_s(v) + \pi_t(v). \end{aligned}$$

If we now proceed as in Dijkstra's algorithm but order the nodes in $\mathcal{Q}$ with respect to f as opposed to merely ordering by d_s, then we obtain Algorithm 2.

Observe that $\pi_t(\cdot) \equiv 0$ is also an underestimator for any graph with non-negative arc lengths, and for this choice of π_t, Algorithm 2 is equivalent to Dijkstra's algorithm.

Algorithm 2: A* search

Input : Graph $G = (V, A)$,
Cost function $d\colon A \to \mathbb{R}_0^+$
Output: Shortest path distance $f(t) = d_s(t)$
Data : Priority queue $\mathcal{Q}$, set $\mathcal{P}$ of processed nodes,
Potential function $\pi_t\colon V \to \mathbb{R}_0^+$

```
1  Q = {(s, π_t(s))};
2  f(v) = ∞ for all v ∈ V \ {s};
3  while Q ≠ ∅ and t ∉ P do
4      (u, f(u)) = Q.removeMin();
5      d_s(u) = f(u) − π_t(u);
6      for v ∈ N_G^+(u) do
7          if v ∈ P then go to line 6 ;
8          altKey = d_s(u) + d(u, v) + π_t(v);        // alternative key via u
9          if v ∈ Q and f(v) > altKey then
10             Q.update(v,altKey);
11         else if v ∉ Q then
12             Q.insert(v,altKey);
13     P.insert(u);
14 return f(t);
```

Another relation to Dijkstra's algorithm is possible whenever we are given a feasible potential π_t. We can then compute the **reduced cost** $d'(u, v)$ of an arc $(u, v) \in A$ by setting

$$d'(u, v) := d(u, v) - \pi_t(u) + \pi_t(v).$$

Note that since π_t is feasible, $d'(u, v) \geq 0$ for every $(u, v) \in A$. In this case, Algorithm 2 is equivalent to Dijkstra's algorithm with the length map d' instead of d. Hence, from the point of view of the algorithm, there is no difference between running A* with the standard costs or Dijkstra with these reduced costs, however there may be a small computational overhead on the reduced Dijkstra since we have to subtract one more value. Yet, through the above observations, we obtain the next result for free.

Theorem 9. *If π_t is feasible, Algorithm 2 yields a shortest path if it exists.*

Proof. The proof is a direct consequence from Theorem 1 and the fact that A* is equivalent to Dijkstra's algorithm for feasible potentials π_t. □

In [HNR68], the authors show a somewhat stronger theorem, by proving that A* also yields an optimal solution if the weaker condition of admissibility is satisfied. We state their theorem as

Theorem 10. *[HNR68, Theorem 1] If π_t is admissible, then the A* algorithm yields a shortest path if it exists.*

In fact, A* is not only optimal in the sense that it finds the shortest path, but it also processes fewer nodes than Dijkstra's algorithm. Intuitively, it is clear that as the potential π_t becomes better, the set of processed nodes becomes smaller. To formalise, let π_t, π_t' be two admissible potentials. We say π_t' **dominates** π_t if and only if

$$\pi_t'(v) \geq \pi_t(v) \quad \text{for all } v \in V,$$

Note that in particular, this means $\pi_t'(t) = \pi_t(t) = 0$. We will denote this domination by $\pi_t' \geq \pi_t$. As one should expect, domination yields a smaller search space, as is shown in the next theorem.

Theorem 11. *[GH04, Theorem 4.1] Let $G = (V, A)$ be a graph and π_t, π_t' be two admissible potentials such that $\pi_t' \geq \pi_t$. Let $\mathcal{P}, \mathcal{P}'$ denote the sets of processed nodes of their respective A* searches. Then $\mathcal{P}' \subset \mathcal{P}$.*

Proof. The proof closely follows [GH04, Theorem 4.1]. We show that if $\mathcal{P}$ does *not* contain a vertex v, then $\mathcal{P}'$ won't contain v either. Let $P = (s = u_0, u_1, \ldots, u_n = v)$ be a shortest path from s to v. So, assuming $v \notin \mathcal{P}$, we have two cases:

(i) $\text{dist}(s, v) + \pi_t(v) > \text{dist}(s, t) + \pi_t(t)$. Since $\pi_t(t) = 0$, this implies

$$\text{dist}(s, t) < \text{dist}(s, v) + \pi_t(v) \leq \text{dist}(s, v) + \pi_t'(v),$$

which means that in this case, $\mathcal{P}'$ cannot contain v either.

(ii) Alternatively, we can encounter $\text{dist}(s, v) + \pi_t(v) = \text{dist}(s, t) + \pi_t(t)$, where the resulting tie in the priority queue must be broken by some criterion such that v comes after t (else v would have been scanned already). Again, we have

$$\text{dist}(s, t) = \text{dist}(s, v) + \pi_t(v) \leq \text{dist}(s, v) + \pi_t'(v),$$

and if we have equality, v is still processed after t, so $\mathcal{P}'$ cannot contain v.

In both cases, we find that v cannot be contained in $\mathcal{P}'$, hence $\mathcal{P}' \subset \mathcal{P}$ must hold. □

Note that since feasible potentials are in particular admissible, the above result also holds for feasible potentials. Theorem 11 has a very important corollary:

Corollary 12. *Algorithm 2 with an admissible potential π'_t processes no more vertices than Algorithm 1.*

Proof. We apply Theorem 11 to the potentials π'_t and $\pi_t(\cdot) \equiv 0$. In the latter case, the A* search is equivalent to Dijkstra' algorithm, which yields the result. □

The different search spaces are indeed visible. As can be seen in Figure 3.4, the route from Berlin to London exhibits a stark contrast between the search spaces of A* and Dijkstra.

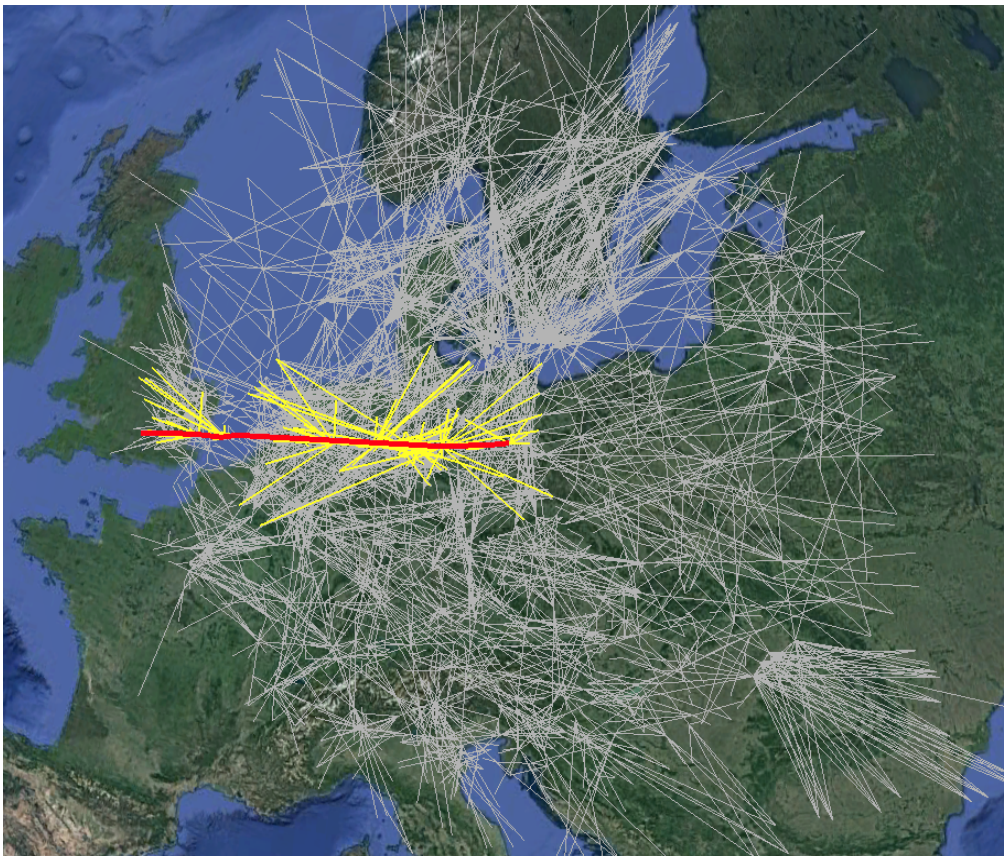


Figure 3.4: Search spaces for static A* (yellow) and Dijkstra's algorithm (gray) on the route TXL to LHR. For visualisation purposes, we show the processed segments rather than the processed nodes (Image data: Google Earth, Digital Globe).

3.2.1 A* in the Time-Dependent Case

Just like we did for Dijkstra's algorithm in Section 3.1.3, we can generalise Algorithm 2 to the time dependent case by keeping track of the tentative time $\tau_{s,\tau_0}(u)$ instead of the

tentative distance $d_s(u)$. Line 8 becomes

$$\texttt{altKey} = \tau_{s,\tau_0}(u) + T((u,v), \tau_{s,\tau_0}(u)) + \pi_t(v),$$

and the subsequent lines are changed accordingly. Of course, the potential function π_t has to be adapted for the time rather than the distance. Analogous to Theorem 9, we obtain the following theorem:

Theorem 13. *Given a* FIFO *network (G, T), any two nodes s, t and a feasible potential π_t, the straightforward generalisation of A* yields an optimal path between s, t on (G, T).*

Proof. Note that feasibility of π_t means that A* is equivalent to Dijkstra's algorithm on (G, T'), where

$$T'((u,v), \tau) = T((u,v), \tau) - \pi_t(u) + \pi_t(v)$$

for every arc $(u, v) \in A$. Hence, by Theorem 6, A* yields an optimal path. □

The main intricacy in the time-dependent case is thus, as in the static case, to come up with a potential – but in the exact time-dependent case of HFTOP, such a function is not straightforward to find. We will go into more detail in Chapter 4.

3.3 Contraction Hierarchies

On a closer inspection of the Airway Network, one can identify a significant number of nodes which have degree two (see also Figure 2.1). While they have a geographical importance, graph-theoretically, these nodes add no information to the network. It makes sense to bypass these unimportant nodes in favour of more important ones, which is one of the observations leading to the algorithm CONTRACTION HIERARCHIES (CHs).

CHs is a fairly new algorithm, developed by Geisberger et al in [GSSV12]. It was introduced to quicken queries on road networks, on which it shows considerable speedup factors over Dijkstra's algorithm, namely up to $\times 41\,000$, as shown in [DSSW09]. Since our network shares some characteristics with road networks, it makes sense to test CHs on our network. Hence, we will in this section motivate and introduce the algorithm in both its static (CHs) and time-dependent (TCHs) version, and justify its correctness.

3.3.1 Contraction Hierarchies in the Static Case

The main idea is to divide the shortest path query into two stages: in the first stage, each node is assigned a hierarchy level (in our case simply a natural number), where higher numbers correspond to more important nodes. According to this order, the nodes are now contracted, which means that bypasses are inserted into the graph. In the second stage, the actual query takes place in form of a bidirectional Dijkstra search, with the addition that in each direction, only nodes with higher rank are examined. We are now going into more detail, starting out with the query side:

The Query

The idea for the Contraction Hierarchies approach stems from the following observation: it makes sense to say that much-travelled roads have 'more important' junctions (e.g. motorway junctions) than small alleys. If we further impose that we can always say which one of two nodes is more important than the other, this is simply an injection

$$\text{rank}\colon V \to \{1, \dots, |V|\},$$

where each node gets assigned its own 'importance' in the network. Now assume that we plan a route on the road network in Germany, say from somewhere in Berlin (s) to somewhere in Stuttgart (t), then we will most certainly drive a substantial time on the motorways A9 and A71. If we were to employ Dijkstra's algorithm, it would, while finding the shortest path, explore each node with a distance in $[0, \text{dist}(s,t)]$ – but there is no advantage in leaving the motorway and taking a detour on smaller roads. Here, we make use of the rank of the nodes: during the query, we will only explore arcs leading to higher-ranked nodes. This works fine as long as $t > s$, which of course need not be the case; and even if it were, there need not be a path from s to t which is strictly increasing in rank. Quite the opposite, in most cases the path will have one highest-ranked node somewhere in the middle (in the road network setting, this corresponds to *the* most important junction, e.g. on a motorway). But since we allow the query algorithm only to explore higher ranked nodes, we need to run the bidirectional version of Dijkstra's algorithm. By forward exploring only arcs leading to higher ranked nodes and by backward exploring only arcs coming from higher ranked nodes (cf Figure 3.5), we aim to leave out a substantial number of unimportant nodes in the network – the arcs which are not considered in the query appear dashed in Figure 3.5.

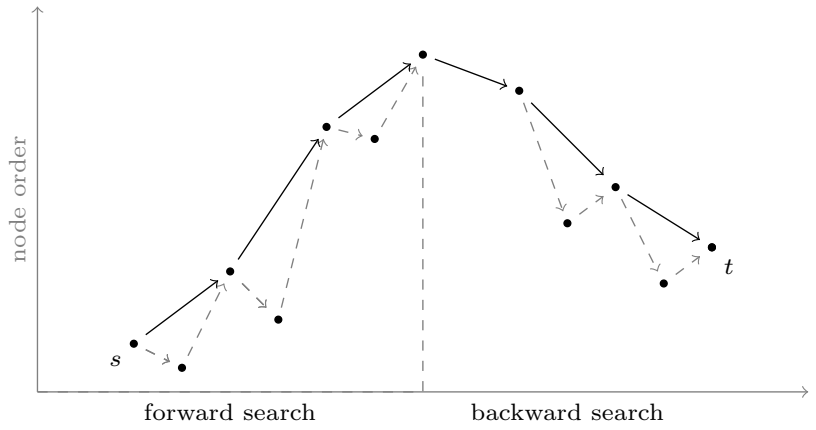


Figure 3.5: Contraction Hierarchies with its forward and backward search space

These nodes are bypassed by shortcut arcs which are added in a preprocessing stage

explained below. Define

$$A^+ = \{(u,v) \in A_{\text{prep}} \colon \text{rank}(u) < \text{rank}(v)\} \text{ and}$$
$$A^- = \{(u,v) \in A_{\text{prep}} \colon \text{rank}(u) > \text{rank}(v)\},$$

where $A_{\text{prep}} \supset A$ is the set of all arcs together with those obtained in preprocessing (see the next subsection). We further call $G^+ = (V, A^+)$ the **upward graph** and $G^- = (V, A^-)$ the **downward graph**. By reverting every arc in A^-, we obtain

$$A^-_{\text{rev}} = \{(u,v) \in A \colon (v,u) \in A^-\},$$

which yields the **search graph** $G^* = (V, \overline{A})$, where $\overline{A} = A^+ \cup A^-_{\text{rev}}$. Analogously to Dijkstra's algorithm, we introduce two priority queues $\mathcal{Q}^+$ and $\mathcal{Q}^-$ for the upward and downward graph respectively. We let r be a marker denoting the current search direction, i.e. $r \in \{+, -\}$ for forward or backward search. Further, let

$$\text{up} \colon A \to \{+, -\}$$

be a function denoting the direction of an arc, i.e. $\text{up}(a) = +$ if and only if $a \in A^+$ and $\text{up}(a) = -$ if and only if $a \in A^-_{\text{rev}}$. The query algorithm is then stated in Algorithm 3.

As with Dijkstra's algorithm or A*, one can obtain a (shortcut) shortest path by keeping track of the predecessor node in each update step. This path can later be unpacked to contain only arcs from A. To speed-up the query, one can prune the search space using the stall-on-demand technique. For a thorough discussion, we refer the reader to [GSSV12].

Preprocessing

To obtain the arcs which ensure that the algorithm neither breaks off unfinished nor that we forfeit correctness, we have to insert certain arcs into the network. Consider the following situation (cf Figure 3.6): the node $v \in V$ has the neighbours u and w with $\text{rank}(u)$, $\text{rank}(w) > \text{rank}(v)$. Suppose that (u, v, w) is the shortest path from u to w. If the query hits the node u, it will at this stage never find w, since we're not allowed to look down. But in preprocessing, we can simply add the arc (u, w) with length $d(u, w) = d(u, v) + d(v, w)$ and the information that it bypasses the node v. Now, when the query arrives at u, it sees the node w and does not explore v, thus saving a step. Notice that we cannot physically delete the arcs (u, v) or (v, w): there may be a query where we arrive at node v, which would then be disconnected from u and w. To formalise, we define the rank-neighbourhoods

$$N^-_{\text{rank}}(v) = \{u \in V \colon \exists\, (u,v) \in A \text{ s.t. } \text{rank}(u) > \text{rank}(v)\} \text{ and}$$
$$N^+_{\text{rank}}(v) = \{w \in V \colon \exists\, (v,w) \in A \text{ s.t. } \text{rank}(w) > \text{rank}(v)\}.$$

Algorithm 3: Query algorithm for CHs

Input : Graph $G = (V, A)$,
Cost function $d\colon A \to \mathbb{R}_0^+$
Output: Shortest path distance d between s, t
Data : Priority queues $\mathcal{Q}^+, \mathcal{Q}^-$

```
1  Q+ = {(s, 0)};
2  Q- = {(t, 0)};
3  r = −;
4  d^r(v) = ∞ for all v ∈ V; d = ∞;
5  d+(s) = 0; d−(t) = 0;
6  while (Q+ ≠ ∅ or Q− ≠ ∅) and (d > min{min Q+, min Q−}) do
7      if Q^{−r} ≠ ∅ then r = −r;
8      (u, d^r(u)) = Q^r.removeMin();
9      d = min{d, d+(u) + d−(u)};
10     for a = (u, v) ∈ Ā do
11         if up(a) = r and (d^r(v) > d^r(u) + d(u, v)) then
12             d^r(v) = d^r(u) + d(u, v);
13             Q^r.update(v, d^r(v));
14 return d;
```

We now go through every pair of nodes $u \in N^-_{\text{rank}}(v)$ and $w \in N^+_{\text{rank}}(v)$ and insert a bypass (or shortcut) arc whenever the path (u, v, w) is the only shortest path from u to w. Through this procedure, we obtain a preprocessed graph $G_{\text{prep}} = (V, A_{\text{prep}})$, where $A_{\text{prep}} \supset A$ consists of all existing arcs and those inserted during preprocessing. To remember which nodes the newly inserted arcs bypassed, we introduce the function

$$b\colon A_{\text{prep}} \setminus A \to V,$$

which maps each a to the node for which we inserted the shortcut. All in all, this yields Algorithm 4.

The output of Algorithm 4 is a preprocessed graph G_{prep}, which we call a **contraction hierarchy**. Note that in line 5 it is important that (u, v, w) is *the only* shortest path, not merely *a* shortest path. If there were another path W using only nodes w' with $\text{rank}(w') > \text{rank}(v)$ such that $\ell((u, v, w)) = \ell(W)$, then no shortcut (u, w) is needed, as we can always travel the same distance by choosing W instead. Such a W is called a **witness path** for (u, v, w). There are further refinement techniques as to how to search for witness paths or how to order the nodes for contraction. We again refer the

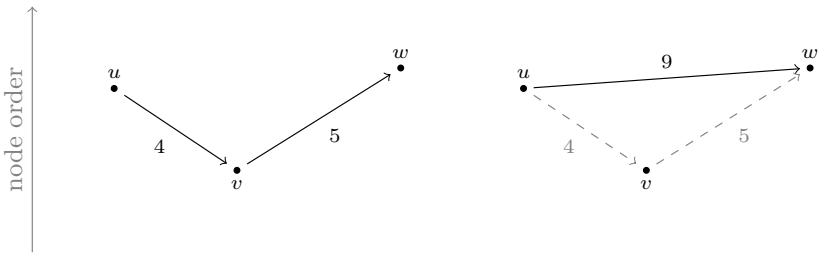


Figure 3.6: Preprocessing in CHs: An arc (u, w) is inserted, bypassing node v

Algorithm 4: Preprocessing for CHs

Input : Graph $G = (V, A)$,
Cost function $d\colon A \to \mathbb{R}_0^+$
Output: Graph $G_{\text{prep}} = (V, A_{\text{prep}})$

```
1 for i = 1, ..., |V| do
2     Find v such that rank(v) = i;
3     for u ∈ N⁻_rank(v) do
4         for w ∈ N⁺_rank(v) do
5             if (u, v, w) is the only shortest path u → w then
6                 Arc a = G.addArc(u,w);
7                 d(a) = d(u, v) + d(v, w);
8                 b(a) = v;
```

reader to [GSSV12] for a more thorough discussion on these, and on means to keep the preprocessed graph sparse.

Optimality of Contraction Hierarchies

With all the nodes left out in the query, it is not trivial that the query algorithm for CHs yields a correct result. We dedicate this subsection to show that this is indeed the case. In our considerations, we follow [GSSV12].

Theorem 14. *[GSSV12, Theorem 1] Given a contraction hierarchy, the Algorithm 3 returns the correct shortest path distance.*

Proof. The proof relies heavily on the fact that Dijkstra's algorithm yields shortest paths, as it is the main ingredient in Algorithm 3. A closer look at the definition of a shortcut shows that the distance between s and t in G^* must be the same as in G. The problem is that Algorithm 3 only finds so-called up-down paths

$$(s = v_0, \ldots, v_p, \ldots, v_n = t),$$

where $\operatorname{rank} v_i < \operatorname{rank} v_{i+1}$ for all $i \in \{0, \dots, p-1\}$ and $\operatorname{rank} v_i > \operatorname{rank} v_{i+1}$ for all $i \in \{p, \dots, n-1\}$. We therefore prove that if there is an s-t-path, then there is also a path of the same length which is an up-down path. Assume now that we are given any shortest s-t-path

$$P = (s = u_0, \dots, u_p, \dots, u_n = t).$$

We define the set M_P of local rank-minima on P as

$$M_P = \{u_i \colon i \in \{1, \dots, n-1\}, \operatorname{rank} u_{i-1}, \operatorname{rank} u_{i+1} > \operatorname{rank} u_i\}.$$

Now if P is not an up-down path, then $M_P \neq \emptyset$. As the function $\operatorname{rank} \colon V \to \{1, \dots, |V|\}$ is an injection, $\operatorname{argmin}_{u \in M_P} \operatorname{rank}(u)$ is a singleton and we can misuse notation to write

$$u_j = \operatorname{argmin}_{u \in M_P} \operatorname{rank}(u).$$

Consider the arcs (u_{j-1}, u_j) and (u_j, u_{j+1}). Both arcs are in A, and exist before u_j is contracted. We have two cases:

(i) There is a witness path $W = (u_{j-1}, w_1, \dots, w_n, u_j)$, with $\operatorname{rank} w_i > \operatorname{rank} u_j$ for all $i \in \{1, \dots, n\}$. Then, since P is a shortest path, we find $\ell(W) = d(u_{j-1}, u_j) + d(u_j, u_{j+1})$, in which case we can replace (u_{j-1}, u_j, u_{j+1}) by W.

(ii) There is no such witness path. In that case, during the preprocessing we will have added a shortcut edge $a = (u_{j-1}, u_{j+1})$ with the weight $d(u_{j-1}, u_j) + d(u_j, u_{j+1})$.

In both cases, we find that we can bridge the valley by either W or a, and obtain a path $P^{(1)}$ with $\ell(P^{(1)}) = \ell(P)$. Further, we can eliminate u_j from M_P. Iterating the above yields a path $P^* = P^{(|M|)}$ which has the same length as P and satisfies $M_{P^*} = \emptyset$, which in turn means that P^* is an up-down path. □

3.3.2 Contraction Hierarchies in the Time-Dependent Case

In the static case, preprocessing is rather straightforward. There are, however, some intricacies involved when it comes to the time-dependent case. We shall denote the time-dependent sibling of CHs by Time-dependent Contraction Hierarchies, or TCHs for short. Throughout this whole subsection we will assume that any travel time function is given as a piecewise linear function.

While in the above case, it suffices to run Dijkstra searches on the static graph (G, d) in order to find witness paths, one must now propagate an entire function through the graph, yielding what we call a **profile query**. This query can be implemented by generalising Dijkstra's algorithm to functions[BDSV09]. Assume we are given two arcs (u, v) and (v, w) – instead of the arc lengths $d(u, v)$ and $d(v, w)$, one now uses the TTFs

T_{uv} and T_{vw}. Adding two edge weights corresponds to chaining these TTFs into a new piecewise linear function

$$T_{uw} = T_{vw} \circ T_{uv}$$

containing the information of both. Taking the minimum of two values is replaced by taking the minimum of two travel time functions.

Suppose we are again given a situation as in Figure 3.6, only this time, the arc weights change over time. Whenever there is *some* time τ such that (u, v, w) is a shortest path, we have to introduce a bypass (u, w) for the node v. In practice, Batz et al avoid many of the computational difficulties associated with the profile searches by pre-selecting cases where shortcuts are definitely or not at all necessary. A full description of their techniques can be found in [BDSV09].

Another important difference between the static and the time-dependent case deals with the query rather than preprocessing: we have seen in Algorithm 3 that CHs makes use of a bidirectional Dijkstra to find the shortest path. While in the static case we only consider distances, in the time-dependent case, we have to know the arrival time in advance in order to successfully do a bidirectional search. As this is clearly not possible, in practice one follows a different approach: in the backward search, one explores all nodes that can reach the target node in G^-. This yields a set of explored arcs A^{exp}, and one then stages a unidirectional search on $A^+ \cup A^{\text{exp}}$. This query algorithm of TCHs also yields an optimal result, if the preprocessing is modified to the travel time functions as above. We quote the result as

Theorem 15. *[BGS08, Theorem 1] The query algorithm of* TCH*s yields the shortest path length.*

The proof is, as the authors state, very similar to the proof of 14, which is why we omit it here.

4 Algorithms for HFTOP

In this chapter, we describe our contributions to HFTOP. We start out with the static case where no wind is blowing, and consider all of the three algorithms introduced in Chapter 3, before leading into the time-dependent case. As we prove a criterion to verify whether the FIFO property holds, we show that HFTOP can be solved to optimality by Dijkstra's algorithm and its derived variants. In Section 3.2, we have shown how A* uses a potential function to guide its search; we therefore introduce a specific potential function based on Super-Optimal Wind for the time-dependent A*. As a potential function must be an underestimator, we prove that Super-Optimal Wind can be used to underestimate the travel time, which leads to a feasible potential function. Further, we assess the quality of the estimation. The main results of this chapter will be published in [BBH+16].

4.1 The Static Case

According to Section 3.1.1, Dijkstra's Algorithm solves SPP optimally whenever we have non-negative arc costs. For these costs, we use the **great circle distance** (GCD), which is always non-negative. Consequently, Dijkstra's algorithm will yield an optimal path by Theorem 1, as will A* with an admissible potential (see Theorem 10). Thus, the only intricacy is to find a good potential function.

To this end, assume we are given a target airport $t \in V$. Then we can use the GCD from any node $v \in V$ to t as underestimator:

$$\begin{aligned} \pi_t \colon V &\to \mathbb{R}_0^+ \\ v &\mapsto \operatorname{gcd}(v, t) \end{aligned}$$

That π_t is an underestimator follows from the triangle-inequality and the fact that a segment's length is defined as the great circle distance and we always fly at least as much as the GCD. In particular, π_t is an admissible potential on the Airway Netwok. Here, choosing the GCD often leads to a tight underestimation, because routes in the graph tend to follow the geodesic. On a road network, however, the great circle distance might be a poor choice due to a more grid-like topology (e.g. the Manhattan street layout in New York), and the occurence of natural obstacles such as rivers or mountain ranges.

A further advantage of using the GCD as an underestimator is that it can be computed at runtime, which means that, in the static case, A* needs no preprocessing. The

evaluation of the great circle distance is rather costly, amounting to up to half of the runtime of the statistics listed in Chapter 5.

4.2 The Exact Time Dependent Case

All three of the algorithms introduced in Section 3 require that our problem satisfy the FIFO property. Due to the nature of the travel time formula (2.1), this is not always the case. Nevertheless, we show that when the maximum wind speed is within reasonable bounds and a certain criterion is satisfied, we can indeed guarantee the FIFO property for our network. In the following, let

$$r_a(\tau) = \|\vec{w}(a, \tau)\|$$

denote the wind speed, and let

$$r^* := \max_{\tau \in [t_0, t_r], a \in A} r_a(\tau)$$

denote the maximum wind speed over all times and arcs. We will in the following theorem set up a sufficient condition for our network to satisfy the FIFO property.

Theorem 16. *Assume that $cr^* \leq v_A$ for some constant $c \geq 1$. Then if for every arc $a \in A$ the variations $w'_C(\tau)$ and $w'_T(\tau)$ of crosswind and trackwind satisfy*

$$\frac{|w'_C(\tau)|}{\sqrt{c^2 - 1}} + w'_T(\tau) \leq \frac{v_A^2}{d_G^a} \left(1 - \frac{1}{c}\right)^2,$$

the family of functions

$$\begin{aligned} T_a \colon [0, \infty) &\to [0, \infty) \\ \tau &\mapsto \frac{d_G^a}{v_G(a, \tau)} \end{aligned}$$

as defined in (2.1) *satisfies the* FIFO *property.*

Proof. Fix an arc $a \in A$, let $\tau \in [t_0, t_r]$ be some time, and $\Delta\tau > 0$. Recall that the FIFO principle states that (see (3.1))

$$\tau + T_a(\tau) \leq (\tau + \Delta\tau) + T_a(\tau + \Delta\tau).$$

We shall in the following write $T(\tau) := T_a(\tau)$, $d_G := d_G^a$ and $v_G(\tau) := v_G(a, \tau)$ to ease notation. The above is then equivalent to

$$\frac{T(\tau + \Delta\tau) - T(\tau)}{\Delta\tau} \geq -1,$$

so, if we can prove that $T'(\tau) \geq -1$ for all τ, then we are done. By (2.1), we have

$$T'(\tau) = -\frac{d_G v'_G(\tau)}{v_G^2(\tau)},$$

which means we have to prove that

$$v'_G(\tau) \leq \frac{v_G^2(\tau)}{d_G}. \tag{4.1}$$

We will first concentrate on lower-bounding the right hand side of this inequality. We claim that

$$v_G(\tau) = \sqrt{v_A^2 - w_C^2(\tau)} + w_T(\tau) \geq v_A - r^* \tag{4.2}$$

holds, i.e. we can always lower-bound the ground speed by $v_A - r^*$. Clearly, the claim is true if and only if

$$v_A^2 - w_C^2(\tau) \geq (v_A - (r^* + w_T(\tau)))^2 = v_A^2 - 2v_A(r^* + w_T(\tau)) + (r^* + w_T(\tau))^2,$$

which in turn is equivalent to

$$2v_A(r^* + w_T(\tau)) \geq (r^*)^2 + 2r^* w_T(\tau) + (w_T^2(\tau) + w_C^2(\tau)) = (r^*)^2 + 2r^* w_T(\tau) + r^2(t).$$

Note that if we substitute $(r^*)^2 \geq r^2(\tau)$ in the very last summand and the claim still holds, it will a fortiori be true for all of the above. So, we obtain

$$v_A(r^* + w_T(\tau)) \geq r^*(r^* + w_T(\tau)),$$

which is always the case since $w_T(\tau) + r^* \geq 0$ and because of the assumption in the statement. Hence, (4.2) holds. Squaring its right hand side yields

$$(v_A - r^*)^2 \geq \left(v_A - \frac{v_A}{c}\right)^2 = v_A^2 \left(1 - \frac{1}{c}\right)^2, \tag{4.3}$$

and combining (4.2) and (4.3) leads to

$$\frac{v_A^2}{d_G}\left(1 - \frac{1}{c}\right)^2 \leq \frac{v_G^2(\tau)}{d_G}. \tag{4.4}$$

Let us now consider the left hand side of the inequality (4.1). The definition of v_G yields that

$$v'_G(\tau) = \frac{\mathrm{d}}{\mathrm{d}\tau}\left(\sqrt{v_A^2 - w_C^2(\tau)} + w_T(\tau)\right) = \frac{-2w'_C(\tau)w_C(\tau)}{2\sqrt{v_A^2 - w_C^2(\tau)}} + w'_T(\tau).$$

which means that we find (again with $cr^* \leq v_A$)

$$\begin{aligned} v'_G(\tau) &\leq \frac{|w'_C(\tau)| \cdot |w_C(\tau)|}{\sqrt{v_A^2 - w_C^2(\tau)}} + w'_T(\tau) \leq \frac{|w'_C(\tau)| \cdot r^*}{\sqrt{v_A^2 - (r^*)^2}} + w'_T(\tau) \\ &\leq |w'_C(\tau)| \cdot \frac{r^*}{\sqrt{(cr^*)^2 - (r^*)^2}} + w'_T(\tau) = \frac{|w'_C(\tau)|}{\sqrt{c^2-1}} + w'_T(\tau). \end{aligned} \tag{4.5}$$

So, together with the assumption from the statement of the theorem, we obtain the chain of inequalities

$$v'_G(\tau) \underset{(4.5)}{\leq} \frac{|w'_C(\tau)|}{\sqrt{c^2-1}} + w'_T(\tau) \leq \frac{v_A^2}{d_G}\left(1 - \frac{1}{c}\right) \underset{(4.4)}{\leq} \frac{v_G^2(\tau)}{d_G}.$$

Note that this proves (4.1), and thus, the claim. □

The above gives us a sufficient condition to decide whether our input satisfies the FIFO property. Within our application, we can refine this criterion: recall that we are given weather prognoses which are three hours apart, and that we linearly interpolate both wind speed and direction in between. So, if we fix an arc $a \in A$ and assume a discretisation $t_0 < t_1 < \ldots < t_r$, let $\tau \in [t_i, t_{i+1}]$ and $\lambda(\tau) = \dfrac{\tau - t_i}{t_{i+1} - t_i}$, then define

$$r(\lambda(\tau)) = r_{i+1} + \lambda(\tau)(r_i - r_{i+1}) \quad \text{and} \quad \theta(\lambda(\tau)) = \theta_{i+1} + \lambda(\tau)(\theta_i - \theta_{i+1}),$$

where $r_j := r_a(\tau_j)$ is the wind speed on a at time τ_j and $\theta_j := \theta_a(\tau_j)$ is its angle. Since $\lambda\colon [t_i, t_{i+1}] \to [0,1]$ is a linear function, its derivative is easy to determine, as are

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}\tau} r(\lambda(\tau)) &= \lambda'(\tau)(r_{i+1} - r_i) = \frac{r_{i+1} - r_i}{t_{i+1} - t_i} \quad \text{and} \\ \frac{\mathrm{d}}{\mathrm{d}\tau} \theta(\lambda(\tau)) &= \lambda'(\tau)(\theta_{i+1} - \theta_i) = \frac{\theta_{i+1} - \theta_i}{t_{i+1} - t_i}. \end{aligned}$$

Applied to the formula (2.2), we find

$$w'_C(\tau) = \frac{r_{i+1} - r_i}{t_{i+1} - t_i} \cdot \sin\theta(\lambda(\tau)) + r(\lambda(\tau)) \cos\theta(\lambda(\tau)) \cdot \frac{\theta_{i+1} - \theta_i}{t_{i+1} - t_i}.$$

Clearly, $r(\lambda(\tau))$ is dominated by the greatest overall wind speed r_a^* on a. As discussed in Section 2.3.2, we always interpolate the wind direction via the smaller of both possible angles, i.e., $|\theta_{i+1} - \theta_i| \leq \pi$ always holds. Further, notice that

$$\begin{aligned} \sin(x) + b\cos(x) &= \sqrt{1+b^2} \cdot \left(\sin(x)\frac{1}{\sqrt{1+b^2}} + \cos(x)\frac{b}{\sqrt{1+b^2}}\right) \\ &= \sqrt{1+b^2} \cdot \Big(\sin(x)\cos(\arctan(b)) + \cos(x)\sin(\arctan(b))\Big) \\ &= \sqrt{1+b^2} \cdot \sin(x + \arctan(b)), \end{aligned}$$

which yields (with $b = \pi$)

$$\begin{aligned}
|w'_C(\tau)| &\leq \frac{|r_{i+1} - r_i| \sin\theta(\lambda(\tau)) + r_a^*|\theta_{i+1} - \theta_i| \cos\theta(\lambda(\tau))}{t_{i+1} - t_i} \\
&\leq \frac{r_a^*(\sin\theta(\lambda(\tau)) + \pi\cos\theta(\lambda(\tau)))}{t_{i+1} - t_i} \\
&\leq \frac{r_a^*\sqrt{1+\pi^2} \cdot \sin(\theta(\lambda(\tau)) + \arctan(\pi))}{t_{i+1} - t_i} \\
&\leq \frac{r_a^*\sqrt{1+\pi^2}}{t_{i+1} - t_i}.
\end{aligned}$$

Similar considerations for (2.3) lead to

$$w'_T(\tau) \leq \frac{r_a^*\sqrt{1+\pi^2}}{t_{i+1} - t_i},$$

and we can upper-bound the sum $\frac{|w'_C(\tau)|}{\sqrt{c^2-1}} + w'_T(\tau)$ by

$$\frac{|w'_C(\tau)|}{\sqrt{c^2-1}} + w'_T(\tau) \leq \left(1 + \frac{1}{\sqrt{c^2-1}}\right) \cdot \frac{r_a^*\sqrt{1+\pi^2}}{t_{i+1} - t_i}.$$

Hence, it is in our application enough to check whether

$$r_a^* \leq \frac{v_A^2}{d_G^a} \left(\frac{\left(1 - \frac{1}{c}\right)^2}{1 + \frac{1}{\sqrt{c^2-1}}} \right) \cdot \frac{t_{i+1} - t_i}{\sqrt{1+\pi^2}}$$

holds for every arc $a \in A$. For better readability, we write

$$C(c) := \left(1 - \frac{1}{c}\right)^2 \cdot \frac{1}{1 + \frac{1}{\sqrt{c^2-1}}} = \left(1 - \frac{1}{c}\right)^2 \cdot \frac{\sqrt{c^2-1}}{1 + \sqrt{c^2-1}}.$$

and find that we have just proved

Corollary 17. *Suppose the wind data is obtained by evaluating only at the points in $\mathcal{T} = \{t_0, \ldots, t_r\}$ and interpolated linearly in between. Assume further that $cr^* \leq v_A$ for some constant $c \geq 1$. Then if for every arc $a \in A$ we have*

$$r_a^* \leq \frac{v_A^2}{d_G^a} \cdot C(c) \cdot \frac{t_{i+1} - t_i}{\sqrt{1+\pi^2}},$$

the travel time functions as in (2.1) satisfy the FIFO *property.*

The assumptions and criteria in Theorem 16 and in Corollary 17 might seem rather restrictive, but in practice, we can always assume that the wind speed is less than half the airspeed of a commercial airliner, hence the assumption $cr^* \leq v_A$ for some $c \geq 1$ is more than justified – an aircraft's airspeed is typically 230 m/s, the wind speed would have to be stronger than that to violate the assumption, which is very unlikely – wind speeds above 33 m/s are already considered as hurricane force.[1] Indeed, our experiments show that c seems to be around $c \approx 3$. In the following, we illustrate that the criterion as in 17 is satisfied by lower and upper bounds for v_A and d_G respectively.

Assume an aircraft flies with an airspeed of 230 m/s, and say it enters a rather long segment $a \in A$, with 1000 km length. As usual for weather prognoses, we assume t_i and t_{i+1} to be three hours apart. Then, with a conservative $c = 2.1$, we find that

$$r_a^* \leq \frac{230^2}{1\,000\,000} \cdot C(2.1) \cdot \frac{10800}{\sqrt{1+\pi^2}} \frac{\text{m}}{\text{s}} \approx 27.5 \frac{\text{m}}{\text{s}},$$

so the maximum wind speed on a must not be greater than 27.5 m/s. To put this into perspective: 27.5 m/s correspond to Beaufort scale 10 wind, a storm.

We can also examine the inequality from the other direction: because of $r_a^* \leq \frac{v_A}{c}$, we can check for which d_G the inequality

$$\frac{v_A}{c} \leq \frac{v_A^2}{d_G} \cdot C(c) \cdot \frac{10800}{\sqrt{1+\pi^2}}$$

holds true. Assuming a conservative $c = 1.5$, this yields

$$d_G \leq c \cdot v_A \cdot C(c) \cdot \frac{10800}{\sqrt{1+\pi^2}} \approx 66.3\,\text{km},$$

so if the wind was bounded by two thirds of the airspeed, the maximum segment length such that the criterion from Corollary 17 is still valid, is 66.3 km. However, this is a very pessimistic estimation and unlikely to occur in practice.

We checked a sample of the instances described in Section 5.1 to see whether they satisfied the criterion in Theorem 16, and found it to hold for almost all segments, with less than 0.02% of them violating it. Note that this does not automatically mean that these instances of HFTOP do not satisfy the FIFO property, as the criterion set up in Theorem 16 is merely a sufficient condition. When this condition is violated, one can still check whether equation (3.1) holds, which we found to be true for those cases. Notice that even in the case that (3.1) was violated, we could render the network FIFO again, by simply subdividing the segments where the FIFO property does not hold into smaller segments and evaluating the travel time for each sub-segment.

[1] This holds true for winds on the ground. High altitude winds can be stronger, but practically never match the airspeed of an aircraft.

Together, Theorem 16 and Corollary 17 guarantee that both Dijkstra's Algorithm and A* yield optimal results in the time-dependent case. We are therefore able to obtain optimal solutions for HFTOP in polynomial time.

Note that while the travel time functions satisfy the FIFO property in our case, in the more general FTOP setting this may not be true. Even if we assume the airspeed to be constant in FTOP, there are still far too many possible problems: firstly, if we allow aircraft to climb or to descend, we can no longer guarantee the FIFO property. This is due to the fact that now the weight of an aircraft becomes important, as lighter aircraft climb faster than heavier aircraft. Thus, a formerly heavier aircraft can, after a CLIMB phase, end up lighter than a formerly lighter one, an the FIFO property must be adapted to respect the weight change. In our case, no such adaptation is necessary, because travel time and fuel consumption are equivalent in the CRUISE phase.

Secondly, introducing restrictions such as NOTAMs (Notice to Airmen) further complicate the matter, as these restrictions can be linked to a time interval. In this case, an earlier aircraft may be prevented from entering the segment altogether and has to circumnavigate (one can view this segment as having ∞ cost in said time interval), while the later aircraft can overtake the first one through the now valid segment.

4.2.1 The Super-Optimal Wind Potential Function

The challenge for A* in the time dependent case is now to find a good potential. In the following, we aim to find a function $\pi_t \colon V \to \mathbb{R}_0^+$ which always underestimates the actual travel time. If we imagine a situation where an aircraft follows a jet stream, it becomes clear why the formerly used GCD no longer underestimates the actual distance: we now have to consider the wind-dependent air distance, which can be much shorter than the GCD. We will therefore use that we are given the weather prognoses for the near future, and aim to guide A* employing this information.

As opposed to road networks, we have another advantage: we know a priori that there is a very limited number of target airports – only roughly 1 300 are used by major airlines.[2]. We can use this fact to our advantage by precomputing a lower bound on the travel time from every waypoint to each target airport. This approach bears strong similarities to the ALT algorithm[GH04], however we note that for an s-t query, we only use one potential function, namely π_t (as opposed to a set of landmarks in ALT).

To precompute the lower bound on the travel time, assume we are given the target airport t. We wish to compute the minimum travel time from each node v to t, which involves finding the minimum travel time on each arc. However, obtaining this minimum is not straightforward, since the travel time function is non-linear (cf Figure 2.4). In the following, we shall therefore explain how to underestimate the travel time from any node v to t, rather than compute the minimum directly. Let us for now fix an arc

[2]Data taken from flightradar24.com

$a \in A$. As becomes apparent from (2.1), underestimating the travel time is equivalent to overestimating the ground speed $v_G(a, \cdot)$. We shall hence devote the remainder of the section toward overestimating the ground speed.

Intuitively, the ground speed of an aircraft is greater, the more trackwind it experiences and the less crosswind there is. We use this intuition for the design of an artificial Super-Optimal Wind vector which is always at least as good as the prevailing wind conditions.

Formally, let $v_G^*(a) = \max_{\tau \in [t_0, t_r]} v_G(a, \tau)$. Let $t_0 = \tau_0 < \tau_1 < \ldots < \tau_N = t_r$ be a discretisation of $[t_0, t_r]$ such that $\tau_i - \tau_{i-1} = \Delta$ for all $i = 1, \ldots, n$. We choose N such that $r|N$, in order to make sure that for all $i = 0, \ldots, n-1$, we always find some $j = 0, \ldots, n-1$ such that $[\tau_i, \tau_{i+1}] \subset [t_j, t_{j+1}]$. We then define for $i = 1, \ldots, n$

$$\underline{w}_C^{(i)}(a) = \min_{\tau \in [\tau_{i-1}, \tau_i]} |w_C(a, \tau)| \text{ and}$$
$$\overline{w}_T^{(i)}(a) = \max_{\tau \in [\tau_{i-1}, \tau_i]} w_T(a, \tau),$$

that is, the minimum crosswind and maximum trackwind on each discretisation step. These can then be used to compute the overestimated ground speed on each interval, denoted by

$$\overline{v}_G^{(i)}(a) = \sqrt{v_A^2 - \left(\underline{w}_C^{(i)}(a)\right)^2} + \overline{w}_T^{(i)}(a).$$

The vector $w_{\text{opt}}^{(i)}$ defined by its cross- and track components

$$w_{\text{opt}}^{(i)} = (\underline{w}_C^{(i)}, \overline{w}_T^{(i)})$$

is then called the **Super-Optimal Wind vector** on $[\tau_{i-1}, \tau_i]$. Furthermore, we define

$$\overline{v}_G(a) = \max_{i \in \{1, \ldots, n\}} \overline{v}_G^{(i)}(a).$$

The intuition from above turns out to be correct, as $\overline{v}_G(a)$ is indeed an overestimator of $v_G(a, \cdot)$. This is shown in the following

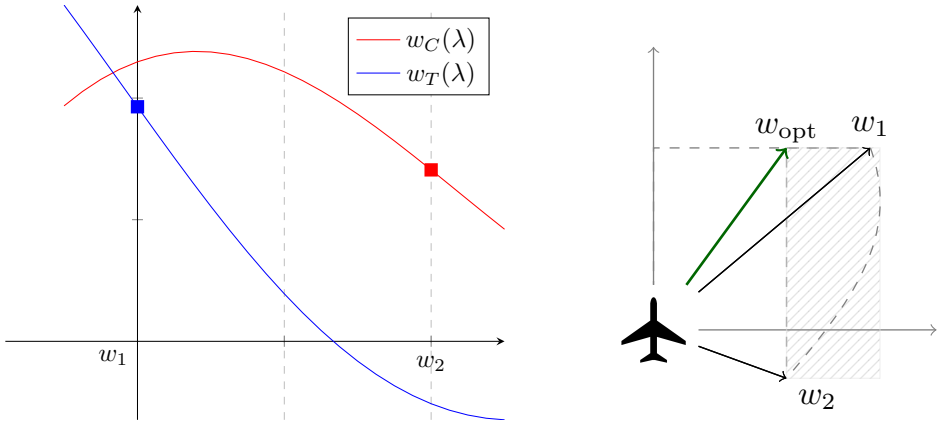
Lemma 18. *We have $v_G(a, \tau) \leq \overline{v}_G(a)$ for all $\tau \in [t_0, t_r]$.*

Proof. Choose an interval $I_k = [t_{k-1}, t_k]$. Since $\underline{w}_C^{(k)}(a) \leq w_C(a, \tau)$ and $\overline{w}_T^{(k)}(a) \geq w_T(a, \tau)$ for any $\tau \in [t_{k-1}, t_k]$, we find

$$v_G(a, \tau)\Big|_{I_k} \leq \max_{\tau \in I_k} \left(\sqrt{v_A^2 - w_C^2(a, \tau)} + w_T(a, \tau) \right) \leq \overline{v}_G^{(k)}(a),$$

As this inequality holds for every I_k, the result follows. □

Since the above lemma guarantees $v_G(a,\tau) \leq \overline{v}_G(a)$, we also find $v_G^*(a) \leq \overline{v}_G(a)$, so $\overline{v}_G(a)$ overestimates the ground speed. Note that $\overline{v}_G$ is an artificial ground speed in the sense that the Super-Optimal Wind vector defining it need not be an interpolation result. Rather, crosswind and trackwind will attain their respective optima at different times. Such a case is depicted in Figure 4.1. Here, it becomes apparent that w_{opt} incorporates both the least possible crosswind and the maximum possible trackwind.



(a) Crosswind (red) and trackwind (blue) functions for w_1, w_2 in **(b)**

(b) The Super-Optimal Wind vector resulting from w_1 and w_2

Figure 4.1: Two wind vectors and their artificial Super-Optimal Wind (green)

Figure 4.1b shows the Super-Optimal Wind vector for the two given wind vectors w_1 and w_2. The dashed arc is the polar interpolation between both, whereas the hatched rectangle shows all vectors that can result from an arbitrary combination of the crosswind component of one and the trackwind component of the other. The example shows a considerable error between w_{opt} and the dashed arc – this is, however, mainly due to the illustration. In practice, w_{opt} tends to lie rather close to the arc, which leads to the following theorem in which we assert the quality of the ground speed overestimation. In it, we will show that its error with respect to the actual optimum is bounded. To this end, we have to assume that the greatest wind speed r_a^* on an arc is less than half the airspeed v_A of the aircraft – as already discussed, in practice, this is always the case.

Theorem 19. *Suppose $v_A \geq 2r_a^*$, where $r_a^* := \max_{\rho\in[t_0,t_N]} r_a(\rho)$ is the greatest overall wind speed on $a \in A$. Then there exists a constant C such that*

$$0 \leq \overline{v}_G(a) - v_G^*(a) \leq C\Delta.$$

Proof. Fix an arc $a \in A$. The first inequality follows directly from Lemma 18, so the only interesting inequality is the second one which guarantees that the error is bounded.

To prove the second inequality, we show that there exists some $C > 0$ such that

$$\max_{\tau \in [\tau_i, \tau_{i+1}]} \left(\overline{v}_G^{(i)}(a) - v_G(a, \tau) \right) \leq C\Delta,$$

which will prove the claim. Let $I = [\tau_i, \tau_{i+1}]$, and consider $\rho_1, \rho_2 \in I \subset [t_k, t_{k+1}]$. Choose $\lambda_1, \lambda_2 \in [0, 1]$ such that

$$\rho_j = \lambda_j t_k + (1 - \lambda_j) t_{k+1} \text{ for } j = 1, 2.$$

We will now bound the trackwind error and the crosswind error separately, starting out with the former: by the mean value theorem, we have

$$\begin{aligned} |w_T(a, \rho_1) - w_T(a, \rho_2)| &\leq |\rho_1 - \rho_2| \max_{\rho \in I} |w_T'(a, \rho)| \\ &= |\rho_1 - \rho_2| \max_{\rho \in I} |r_a'(\rho) \cos \theta_a(\rho) - r_a(\rho) \theta_a'(\rho) \sin \theta_a(\rho)| \\ &\leq \Delta \left(\max_{\rho \in I} |r_a'(\rho)| + r_a^* \max_{\rho \in I} |\theta_a'(\rho)| \right). \end{aligned} \tag{4.6}$$

Recall that we view wind vectors as polar vectors $w_a^{(j)} = (r_a^{(j)}, \theta_a^{(j)})$. As already discussed, we interpolate both wind speed and direction linearly in $[t_k, t_{k+1}]$. In particular, for $\rho \in [t_k, t_{k+1}]$ this yields the constants

$$r_a'(\rho) = \frac{r_a^{(k+1)} - r_a^{(k)}}{t_{k+1} - t_k} \text{ as well as } \theta_a'(\rho) = \frac{\theta_a^{(k+1)} - \theta_a^{(k)}}{t_{k+1} - t_k}.$$

Together with (4.6), we find that

$$|w_T(a, \rho_1) - w_T(a, \rho_2)| \leq \Delta \frac{\left| r_a^{(k+1)} - r_a^{(k)} \right| + r_a^* \left| \theta_a^{(k+1)} - \theta_a^{(k)} \right|}{t_{k+1} - t_k} \leq \Delta \frac{r_a^*(1 + \pi)}{t_{k+1} - t_k}, \tag{4.7}$$

where the last inequality holds because $|r_a^{(k+1)} - r_a^{(k)}| \leq r_a^*$, and $|\theta_a^{(k+1)} - \theta_a^{(k)}| \leq \pi$ as already shown right before Corollary 17. By analogous procedures as for the trackwind, we can bound the crosswind error by the same value, i.e.

$$|w_C(a, \rho_1) - w_C(a, \rho_2)| \leq \Delta \frac{r_a^*(1 + \pi)}{t_{k+1} - t_k}. \tag{4.8}$$

With these preliminaries, we shall now bound the error on the ground speed. To this end, let $\overline{\rho}$ maximise the trackwind, $\underline{\rho}$ minimise the absolute value of the crosswind, and ρ^* maximise the ground speed in I. So, to be precise, we have

$$\begin{aligned} w_C(a, \underline{\rho}) &:= \underline{w}_C^{(i)}(a) = \min_{\tau \in I} |w_C(a, \tau)|, \\ w_T(a, \overline{\rho}) &:= \overline{w}_T^{(i)}(a) = \max_{\tau \in I} w_T(a, \tau), \end{aligned}$$

as well as $\rho^* \in \operatorname{argmax}_{\tau \in I} v_G(a,\tau)$. Then we find

$$\begin{aligned}\overline{v}_G^{(i)}(a) - v_G(a,\tau) &= \\ &= \sqrt{v_A^2 - w_C(a,\underline{\rho})^2} + w_T(a,\overline{\rho}) - \sqrt{v_A^2 - w_C(a,\rho^*)^2} - w_T(a,\rho^*) \\ &= \frac{w_C(a,\rho^*)^2 - w_C(a,\underline{\rho})^2}{\sqrt{v_A^2 - w_C(a,\underline{\rho})^2} + \sqrt{v_A^2 - w_C(a,\rho^*)^2}} + w_T(a,\overline{\rho}) - w_T(a,\rho^*). \qquad (4.9)\end{aligned}$$

Note that $|w_C(a,\tau)| \leq r_a(\tau) \leq r_a^*$ always holds, and that we can upper-bound the numerator of the above expression by letting

$$\begin{aligned}w_C(a,\rho^*)^2 - w_C(a,\underline{\rho})^2 &\leq |w_C(a,\rho^*) + w_C(a,\underline{\rho})| \cdot |w_C(a,\rho^*) - w_C(a,\underline{\rho})| \\ &\overset{(4.8)}{\leq} |w_C(a,\rho^*) + w_C(a,\underline{\rho})| \cdot \Delta \frac{r_a^*(1+\pi)}{t_{k+1} - t_k} \\ &\leq \Big(|w_C(a,\rho^*)| + |w_C(a,\underline{\rho})|\Big) \cdot \Delta \frac{r_a^*(1+\pi)}{t_{k+1} - t_k} \\ &\leq \Big(r_a(\rho^*) + r_a(\underline{\rho})\Big) \cdot \Delta \frac{r_a^*(1+\pi)}{t_{k+1} - t_k} \\ &\leq 2r_a^* \cdot \Delta \frac{r_a^*(1+\pi)}{t_{k+1} - t_k}.\end{aligned}$$

Applying this to (4.9) yields

$$\begin{aligned}\overline{v}_G^{(i)}(a) - v_G(a,\tau) &\leq \frac{2r_a^* \cdot \Delta \frac{r_a^*(1+\pi)}{t_{k+1}-t_k}}{\sqrt{v_A^2 - r(a,\underline{\rho})^2} + \sqrt{v_A^2 - r(a,\rho^*)^2}} + w_T(a,\overline{\rho}) - w_T(a,\rho^*) \\ &\leq \frac{2r_a^* \cdot \Delta \frac{r_a^*(1+\pi)}{t_{k+1}-t_k}}{\sqrt{v_A^2 - r_a^2(\underline{\rho})} + \sqrt{v_A^2 - r_a^2(\rho^*)}} + w_T(a,\overline{\rho}) - w_T(a,\rho^*) \\ &\leq \frac{2r_a^*}{2\sqrt{v_A^2 - (r_a^*)^2}} \cdot \Delta \frac{r_a^*(1+\pi)}{t_{k+1} - t_k} + w_T(a,\overline{\rho}) - w_T(a,\rho^*),\end{aligned}$$

and together with (4.7), we obtain

$$\begin{aligned}\overline{v}_G^{(i)}(a) - v_G(a,\tau) &\leq \frac{r_a^*}{\sqrt{v_A^2 - (r_a^*)^2}} \cdot \Delta \frac{r_a^*(1+\pi)}{t_{k+1} - t_k} + |w_T(a,\overline{\rho}) - w_T(a,\rho^*)| \\ &\overset{(4.7)}{\leq} \Delta \frac{r_a^*(1+\pi)}{t_{k+1} - t_k} \cdot \left(\frac{r_a^*}{\sqrt{v_A^2 - (r_a^*)^2}} + 1 \right).\end{aligned}$$

Now, since we assumed $v_A \geq 2r_a^*$, we find that

$$r_a^* \left(\frac{r_a^*}{\sqrt{v_A^2 - (r_a^*)^2}} + 1 \right) \leq r_a^* \left(\frac{r_a^*}{r_a^* \sqrt{3}} + 1 \right) \leq \frac{v_A}{2}(1+1) = v_A,$$

and thus we have

$$C = \frac{(1+\pi)v_a}{t_{k+1} - t_k},$$

which proves the claim. □

If we define $T_a^* := \min_{\tau \in [t_0, t_r]} T_a(\tau)$ and $\underline{T}(a) = \frac{d_G(a)}{\overline{v}_G(a)}$, we obtain this chapter's main result:

Theorem 20. *Suppose $v_A \geq 2r_a^*$, where $r_a^* := \max_{\rho \in [t_0, t_N]} r_a(\rho)$ is as in Theorem 19. Then there is a constant C' such that for any arc a, we have*

$$0 \leq T_a^* - \underline{T}(a) \leq C' \Delta.$$

Proof. The left inequality is again clear as a direct consequence from Lemma 18, while the right inequality follows directly from Theorem 19. □

We can now define the lower bound cost graph $(G, \underline{T})$, where $G = (V, A)$, with the static arc costs

$$\begin{aligned} &\underline{T} \colon A \to [0, \infty) \\ &(u, v) \mapsto \underline{T}((u, v)) \end{aligned}$$

Given a target node t, we can run all-to-one Dijkstras on $(G, \underline{T})$ to obtain distance values $\underline{\text{dist}}(v, t)$ for every node $v \in V$. Here, we use the underline to emphasise that the values are computed in the lower bound graph $(G, \underline{T})$. This yields a potential function

$$\begin{aligned} &\underline{\pi}_t \colon V \to \mathbb{R}_0^+ \\ &\qquad v \mapsto \underline{\text{dist}}(v, t) = \sum_{i=1}^{n} \underline{T}(u_{i-1}, u_i), \end{aligned}$$

where $P = (v = u_0, u_1, \ldots, u_{n-1}, u_n = t)$ is a shortest path from v to t.

Lemma 21. *The potential function $\underline{\pi}_t$ is feasible in $(G, \underline{T})$.*

Proof. We have to show that for any arc $(v, w) \in A$, the inequality $\underline{\pi}_t(v) \leq \underline{\pi}_t(w) + \underline{T}(v, w)$ holds. For a contradiction, assume the converse, namely that for some $(v, w) \in A$, we find $\underline{\pi}_t(v) > \underline{\pi}_t(w) + \underline{T}(v, w)$. But from the definition of $\underline{\pi}_t$, we find

$$\underline{\text{dist}}(v, t) > \underline{\text{dist}}(w, t) + \underline{T}(v, w).$$

Let now P_v be a shortest path from v to t, and P_w a shortest path from w to t. Seeing as $\underline{\text{dist}}(v, t) = \ell(P_v)$ denotes the length of P_v, this directly contradicts the notion of P_v being a shortest path, since the path obtained by walking from v to w along (v, w) and then continuing on P_w is clearly shorter. □

The preceding lemma now makes sure that A* yields an optimal solution on $(G, \underline{T})$. In fact, the algorithm also returns an optimal solution in (G, T), as is shown in the following Theorem.

Theorem 22. *The potential $\underline{\pi}_t$ is feasible in (G, T).*

Proof. Just observe that

$$\underline{\pi}_t(v) \leq \underline{\pi}_t(w) + \underline{T}(v, w) \leq \underline{\pi}_t(w) + T_{(v,w)}(\tau)$$

holds for all times $\tau \in [t_0, t_r]$ because of Theorem 20. □

The preceding theorem in conjunction with Theorem 13 yields that A* together with the potential function $\underline{\pi}_t$ resulting from the Super-Optimal Wind solves the time-dependent shortest path problem on the Airway Network optimally. In particular, since $\underline{\pi}_t(t) = 0$, the potential function is admissible by Lemma 8. For this reason, Corollary 12 holds, and we visit at most as many vertices as Dijkstra's algorithm, as can be seen in Figure 4.2.

4.2.2 Minimising the Crosswind and Maximising the Trackwind

In the previous subsection, we have introduced the Super-Optimal Wind and shown that it yields an underestimator. As input, we assumed the maximum trackwind and the minimum crosswind for each arc over a specified time period – all that's left now is to computationally determine them.

A natural choice for the discretisation step is $r = N$, i.e, $\tau_i = t_i$. In our application, we are given weather prognoses containing wind conditions at three hour intervals for the near future. Therefore, it makes sense that the discretisation step size $\Delta = \tau_{j+1} - \tau_j = t_{i+1} - t_i$ be three hours. While a choice $r > N$ is of course possible, we shall see in the next subsection that a step size of three hours already yields very good results.

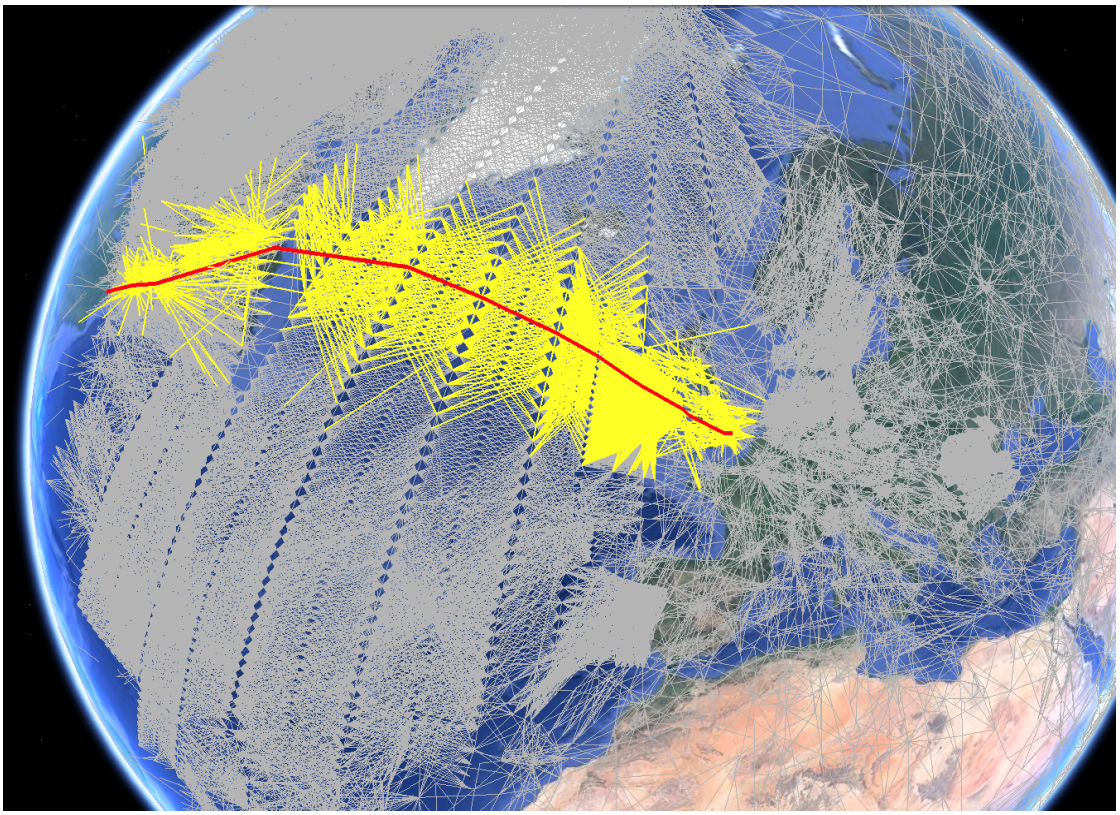


Figure 4.2: Search spaces of Dijkstra's algorithm (gray) and A* (yellow) on a route between London and New York City. The path is shown in red. Again, we visualise the search spaces through the arcs rather than through the waypoints (Image data: Google Earth, Digital Globe).

We will in the following restrict the discussion to the case $i = 1$, as all other cases are similar. Let us fix an arc $a \in A$ and define $\lambda := \dfrac{t_2 - \tau}{t_2 - t_1}$. Recall the formulae for (2.2) and (2.3) as defined in Section 2.3.2:

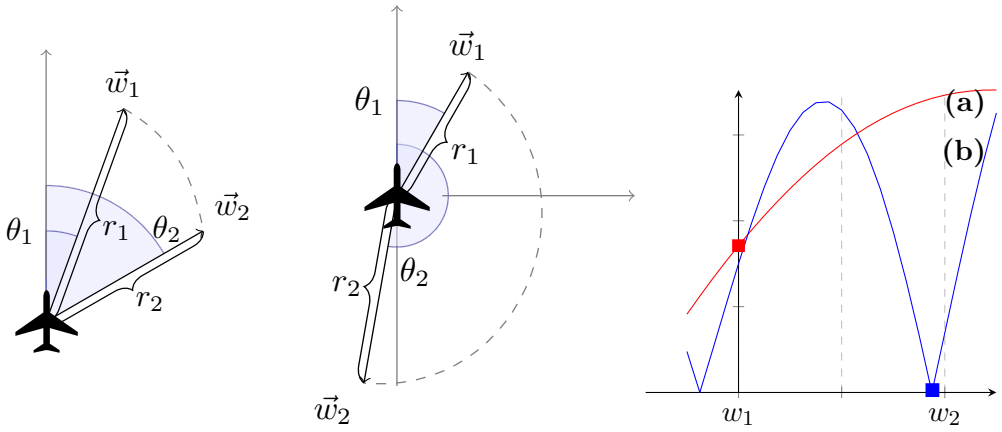
$$w_C(a, \lambda) = (\lambda r_1 + (1 - \lambda) r_2) \sin(\lambda\theta_1 + (1 - \lambda)\theta_2) =: f(\lambda), \text{ and}$$
$$w_T(a, \lambda) = (\lambda r_1 + (1 - \lambda) r_2) \cos(\lambda\theta_1 + (1 - \lambda)\theta_2) =: g(\lambda).$$

To ease notation, we introduce constants $a := r_1 - r_2$, $b := r_2 > 0$, $\alpha := \theta_1 - \theta_2$ and $\beta := \theta_2$. With these, (2.2) and (2.3) become

$$f(\lambda) = (a\lambda + b) \cdot \sin(\alpha\lambda + \beta) \text{ and}$$
$$g(\lambda) = (a\lambda + b) \cdot \cos(\alpha\lambda + \beta).$$

We are first looking to minimise $|w_C(a,\tau)|$ for the given segment a. We may wlog assume that $w_C(a,\tau) \geq 0$ for all $\tau \in [t_1, t_2]$. If this is not the case, then either $w_C(a,\tau) < 0$ for all τ and we can consider $-w_C$. Or, if w_C is both positive and negative, then since it is continuous, the minimum cross wind is zero. To find the minimum when $w_C(a,\tau) \geq 0$, we assume wlog $\theta_1 < \theta_2$ and $r_1 \neq r_2$, as the cases with equal radii or angles are easy.

Although it happens frequently in practice, $\vec{w}_1, \vec{w}_2$ need not lie in the same quadrant. In Figure 4.3b, we have to interpolate the wind vectors through three of the four possible quadrants. Since $[\theta_1, \theta_2]$ is compact, $|w_C|$ attains its minimum: therefore it makes sense to compute the minima in $[\theta_1, \frac{\pi}{2}], [\frac{\pi}{2}, \pi]$ and $[\pi, \theta_2]$ separately and then pick the overall minimum, which is then a minimum in $[\theta_1, \theta_2]$.



(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 4.3: Cases considered for crosswind minimisation

Fortunately, we can easily compute the first three derivatives of f, which are (using the constants defined above)

$$f'(\lambda) = a\sin(\alpha\lambda + \beta) + \alpha(a\lambda + b)\cos(\alpha\lambda + \beta)$$
$$f''(\lambda) = 2a\alpha\cos(\alpha\lambda + \beta) - \alpha^2(a\lambda + b)\sin(\alpha\lambda + \beta).$$
$$f'''(\lambda) = -3a\alpha^2\sin(\alpha\lambda + \beta) - \alpha^2(a\lambda + b)\cos(\alpha\lambda + \beta).$$

Notice that since $\theta_1 < \theta_2$, we have $\alpha < 0$, and we clearly always have $b > 0$ as it is a radius.

The necessary condition for the crosswind function f to have a minimum is that its derivative f' has a root. We shall find this minimum by applying Newton's Method; as a comprehensive introduction of the method would lead us too far off-topic, we refer the reader to [DH08] for an overview.

Since we're only looking in a very narrow interval (namely, $\lambda \in [0,1]$), we can reduce the computational effort by pre-selecting those cases where f' actually has a root in $[0,1]$, thus distinguishing them from cases where we know that the minimum of f must be met at either endpoint of $[0,1]$. To this end, we distinguish several cases:

1. $[\theta_1, \theta_2] \in [0, \frac{\pi}{2}]$. This especially means that $\sin(\alpha\lambda + \beta), \cos(\alpha\lambda + \beta) \geq 0$, and we can open the following two subcases (compare Figures 4.3a and 4.3c)
   1.1 $a > 0$ ($\Leftrightarrow r_1 > r_2$). As $a\lambda + b > 0$ for all λ and since $\alpha < 0$, we know that $f''(\lambda) < 0$ for all $\lambda \in [0,1]$. Hence, f is concave and must meet its minimum in one of $\{0,1\}$.
   1.2 $a < 0$. In this case, f'' can be both positive and negative. A closer look at f''' however reveals that $f'''(\lambda) > 0$ for $\lambda \in [0,1]$, so f'' is monotonously increasing. Now, if $f''(1) \leq 0$, then f is concave within $[0,1]$, and its minimum is, as above, one of the boundary points. If $f''(1) > 0$, we check whether $f''(0) > 0$ (i.e., f is convex), or $f''(0) < 0$. In the latter case, we perform Newton's method to find the inflection point λ_p, the root of f''. We can then write $[0,1] = [0,\lambda_p] \cup [\lambda_p, 1]$ and know that on $[0,\lambda_p]$, f must be concave, and on $[\lambda_p, 1]$, it must be convex. Having done so, we know that the minimum in the concave part is met at either 0 or λ_p.

   To find the minimum when f'' is convex, we again employ Newton's method without any further preparation to find a root of f'. The supposed minimum can be anywhere in $[0,1]$ (or $[\lambda_p, 1]$ for the case just discussed), but if the search runs out of bounds (i.e., it converges towards some point $\lambda^* \notin [0,1]$), then we simply take the $\lambda^* \in \{0,1\}$ which is closest to the supposed root.

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

2. $[\theta_1, \theta_2] \subset [\frac{\pi}{2}, \pi]$. Again we find $\sin(\alpha\lambda + \beta) \geq 0$, but this time $\cos(\alpha\lambda + \beta) \leq 0$, which yields the two subcases:
   2.1 $a > 0$ – as in 1.2, f'' can be both positive and negative. We refer the reader to this case.
   2.2 Similar considerations as in 1.1 yield that $f''(\lambda) < 0$, and hence it is concave.

3. $[\theta_1, \theta_2] \subset [\pi, \frac{3\pi}{2}]$. In this case, we mirror both w_1, w_2 with respect to the track axis and apply 2. This symmetry operation does not distort the results, as we are only considering the crosswind part of w_1, w_2 here, which remains invariant under this mirroring operation.

4. For symmetry reasons, the case $[\theta_1, \theta_2] \subset [\frac{3\pi}{2}, 2\pi]$ is covered by 1.

By the same considerations as before, we can maximise the trackwind function $g(\lambda)$.

4.2.3 Super-Optimal Wind in Practice

We have implemented the Super-Optimal Wind underestimator within the framework of our application. To assess its quality, we ran it on every arc set of all nine instance sets which arise out of three different altitude layers and three different weather conditions described in Section 5.1. To compare, we also implemented an exact (brute force) solver which assumes a discretisation step of one second and computes $v_G(a, \tau)$ for every second. The brute force solver then picks the overall maximum. As our weather prognoses are spaced three hours apart, we chose a discretisation step of three hours for the Super-Optimal Wind. Recall that the computation of the Super-Optimal Wind is solely to obtain a travel-time underestimator – we do not need to obtain the exact result every time, but merely find a good upper bound. Our results show that the seemingly coarse discretisation step of three hours already yields very accurate results: the average error with respect to the brute force solver is $0.434 \cdot 10^{-3}$. Using the same computer as described in Chapter 5, the average time to process an arc is less than one millisecond. On 20 cores, the average runtime for all segments on one layer is 5.61 seconds, and we found that in almost one third of all cases, the Super-Optimal Wind yields the same result as the exact solver.

4.3 Approximating the Travel Time Function

As already discussed in Section 2.3.1, the PWL case promises to yield better query times than the exact case, as we can outsource much of the expensive weather calculation into a preprocessing step and evaluate only the resulting PWLs in the query. Although this possibly leads to errors, the PWL approach is a standard way to model time-dependency on graphs and hence widely used in many time-dependent shortest path algorithms. Again, as in the case of the exact TTFs, we have to make sure that Dijkstra's algorithm yields the optimal result. We do so in Theorem 24, but have to base the result on a small lemma.

Assume we are given a PWL f, and suppose that there exist two points $\tau < \tau'$ which do not satisfy the FIFO property – in other words, $f(\tau) > \tau' - \tau + f(\tau')$, as also depicted in Figure 4.4. It is then intuitively clear that there should be an interval $[t_k, t_{k+1}]$ which also violates the FIFO principle, but the rigorous proof needs some work:

Lemma 23. *Let $f\colon [t_0, t_1] \cup [t_1, t_2] \cup \ldots \cup [t_{r-1}, t_r] \to \mathbb{R}$ be a piecewise linear function, and assume there are $\tau < \tau'$ such that $f(\tau) > \tau' - \tau + f(\tau')$. Then there exists an interval $[t_k, t_{k+1}]$ such that*

$$f(t_k) > t_{k+1} - t_k + f(t_{k+1}).$$

Proof. Say $\tau \in [t_i, t_{i+1}]$ and $\tau' \in [t_j, t_{j+1}]$. We assume that $\tau \neq t_i$ and $\tau' \neq t_j$, or else we can do the proof by skipping the considerations leading to equation (4.12) below. Note

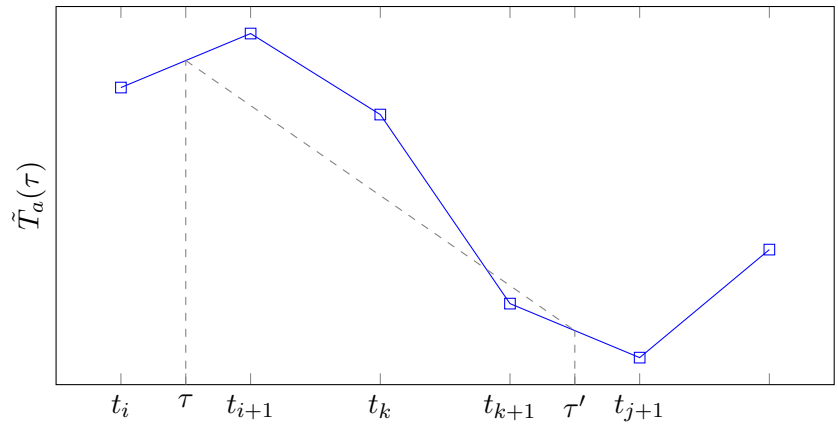


Figure 4.4: A non-FIFO piecewise linear function

that the statement can be rewritten to

$$\frac{f(\tau') - f(\tau)}{\tau' - \tau} < -1. \tag{4.10}$$

We now pick the interval $[t_k, t_{k+1}]$ on which f_k is the most decreasing. To be precise, we choose $k \in \{i, \dots, j\}$ such that

$$\frac{f(t_{k+1}) - f(t_k)}{t_{k+1} - t_k} = \min_{m \in \{i,\dots,j\}} \frac{f(t_{m+1}) - f(t_m)}{t_{m+1} - t_m}.$$

and claim that this is the desired k. Assume for a contradiction that

$$\frac{f(t_k) - f(t_{k+1})}{t_{k+1} - t_k} \geq -1.$$

Using a telescopic sum, we find

$$\frac{f(\tau') - f(\tau)}{\tau' - \tau} = \frac{f(\tau') - f(t_j) + \sum_{m=i}^{j-1} (f(t_{m+1}) - f(t_m)) + f(t_i) - f(\tau)}{\tau' - \tau}. \tag{4.11}$$

We will now treat each summand on its own, starting out with the big sum. Using the

definition of k and the assumption, we find

$$\begin{aligned}\sum_{m=i}^{j-1}(f(t_{m+1})-f(t_m)) &= \sum_{m=i}^{j-1}\frac{f(t_{m+1})-f(t_m)}{t_{m+1}-t_m}\cdot(t_{m+1}-t_m)\\ &\geq \sum_{m=i}^{j-1}\frac{f(t_{k+1})-f(t_k)}{t_{k+1}-t_k}\cdot(t_{m+1}-t_m)\\ &\geq (-1)\cdot\sum_{m=i}^{j-1}(t_{m+1}-t_m)\\ &= (t_i-t_j).\end{aligned}$$

We now look at the last summand, $f(t_i)-f(\tau)=\dfrac{f(t_i)-f(\tau)}{t_i-\tau}(t_i-\tau)$. For $\tau\in[t_i,t_{i+1}]$, an easy calculation shows that

$$\frac{f(t_i)-f(\tau)}{t_i-\tau}=\frac{f(t_i)-f(t_{i+1})}{t_i-t_{i+1}},$$

in other words, the slope does not change on $[t_i,t_{i+1}]$. But this means

$$f(t_i)-f(\tau)=\frac{f(t_i)-f(\tau)}{t_i-\tau}(t_i-\tau)=\frac{f(t_i)-f(t_{i+1})}{t_i-t_{i+1}}(t_i-\tau)\geq(-1)\cdot(t_i-\tau), \tag{4.12}$$

and analogously, we find $f(\tau')-f(t_j)\geq(-1)\cdot(\tau'-t_j)$. Applying these results to (4.11), we have

$$\frac{f(\tau')-f(\tau)}{\tau'-\tau}\geq\frac{-(\tau'-t_j)+(t_i-t_j)-(t_i-\tau)}{\tau'-\tau}=-\frac{\tau'-\tau}{\tau'-\tau}=-1,$$

contradicting (4.10) and thus proving the claim. □

Having proved the lemma, we can now proceed to

Theorem 24. *If the family T_a of travel time functions satisfies the* FIFO *property, then the family of piecewise linear functions*

$$\tilde{T}_a\colon[0,\infty)\to[0,\infty)$$

as defined in (2.4) satisfies the FIFO *property.*

Proof. Suppose not. Then, there exist two time points $\tau<\tau'$ such that

$$\tilde{T}_a(\tau)>(\tau'-\tau)+\tilde{T}_a(\tau').$$

We may by Lemma 23 assume that $\tau = \tau_k, \tau' = \tau_{k+1}$ for some $k \in \{0, \dots, n-1\}$. Notice that since $\tilde{T}_a(\tau_i) = T_a(\tau_i)$ for all i, we then have

$$T_a(\tau_k) > (\tau_{k+1} - \tau_k) + T_a(\tau_{k+1}),$$

so T_a would have to violate the FIFO property contradicting the assumption. □

Theorem 24 states precisely that Dijkstra's algorithm, A* and CHs will yield an optimal path in the approximated time-dependent setting.

In order to put A* to work, we now proceed as in Section 4.2: first, for any arc $a \in A$ we compute a value $\underline{\tilde{T}}_a$ which is a lower bound on all travel times for this particular arc, i.e. we determine $\underline{\tilde{T}}_a$ such that

$$\underline{\tilde{T}}_a \leq \tilde{T}_a(\tau) \text{ for all } \tau \in [t_0, t_r].$$

Since we are given PWLs as travel time functions, finding the minimum is easy: it can be achieved in linear time by going through all the break points and picking the minimal value. Doing so for every arc $a \in A$ yields a lower bound cost graph $(G, \underline{\tilde{T}})$ with the time-independent arc length function $\underline{\tilde{T}}\colon A \to [0, \infty)$, on which we can now run an all-to-one Dijkstra for every target node t. For each such t, any node v that was reached in the all-to-one search will then be assigned the minimum travel time it takes to travel from v to t on $(G, \underline{\tilde{T}})$, which is by extension a lower bound on the travel time in $(G, \tilde{T})$. By analogous means as in Corollary 22, we obtain that the resulting potential function is again feasible in $(G, \tilde{T}_a)$.

5 Computational Results

To assess the quality of the introduced algorithms, we implemented both Dijkstra's algorithm and A* in C++ within the framework of our application, using the potential functions described in Sections 4.2 and 4.3. For CHs and TCHs, we used the tools *Contraction Hierarchies* and *KaTCH* released by the Karlsruhe Institute of Technology KIT[Kar16] (also implemented in C++).

All of our computations were carried out on computers with 132GB of RAM and an Intel(R) Xeon(R) CPU E5-2660 v3 processor with 2.6GHz and 25.6MB cache.

Every preprocessing step was done in parallel and using 20 threads, with the exception of the tool Contraction Hierarchies, whose code does not offer the option of parallelisation. All queries, on the other hand, were done in single-thread mode.

5.1 Instances

Since the Airway Network in our perspective consists of multiple disconnected layers, we can easily generate new instances by switching layers; for that reason, we chose the altitudes 29 000ft, 34 000ft, and 39 000ft. Following aviation convention, we will denote these by FL290, FL340 and FL390 respectively ('FL' for flight level). These three flight levels, while being common cruise altitudes for aircraft, are spaced sufficiently far apart such as to allow for substantially different wind conditions. Although the layers are topologically very similar to each other, there are several segments which are only allowed on some altitudes, thus yielding 329 442, 329 736 and 329 580 arcs, respectively.

We also picked three of the weather prognoses which are available to us, namely December, February and March (denoted by *Dec, Feb, Mar* respectively). These weather prognoses are long enough apart to guarantee independent weather. Each prognosis consists of a set of time points $\{t_0, \ldots, t_r\}$ together with wind conditions at each t_i for a three-dimensional grid spanning the Earth's atmosphere. The t_i cover ranges from 30 through 45 hours, and are all spaced three hours apart. The graph and weather prognoses are based on real world data, provided to us by Lufthansa Systems. As a test set, we used 18644 **origin-destination** (OD) **pairs**, consisting of all flights recorded in June 2015 by the website flightradar24.com.

Any variant of Dijkstra's algorithm will in the following tables be denoted by DIJK, and the A* algorithm can be found labeled as such. Time-dependent Contraction Hierarchies will be labelled as TCHs, and its static counterpart will be referenced to as CHs. All of

them will carry tokens indicating the considered version, e.g. 'PWL' in the index will mean that piecewise linear functions were used for the travel time on the arcs. The index 'E' will denote the exact version of the travel time computation. If they bear no token, they represent the static case.

5.2 Results in the Static Case

Since in the static case we do not consider any wind, but only the great circle distance as length of a segment, we denote each of the instances by I- followed by the first two numbers denoting their altitude, i.e. we have the instances I-29, I-34 and I-39. Since the graph topology does not change much over the layers, one should expect that the query times do not vary much among the instances.

Instance	Dijk	CHs			A*		
	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

Table 5.1: Comparison of CHs and A* for the case of static arc costs.

In Table 5.1, we provide the preprocessing times of CHs and A* and average query times. The latter ones were obtained by averaging the time it took to compute a shortest path for all 18644 OD pairs. As expected, our computations show that the query times are stable for all three instances. Moreover, A* performs slightly better than CHs; however, not enough to be of any significance. Note that in this static case, CHs yields a very low speedup compared to instances on road networks. We identify two main reasons for this behaviour: first, the average degree in the Airway Network is higher than in road networks. While road networks tend to have average degrees between two and three, the average degree in our network is around 12.5, impacting CHs preprocessing. The second reason is the number of arcs in a shortest path. While routes in road networks can easily reach more than one thousand arcs per path for continental-sized networks, routes in the Airway Network consist of only very few edges, averaging on 20 edges per path. CHs' full strength only shows when there are many nodes to bypass, but since routes in the Airway Network are rather short, one cannot attain a good speedup here.

Moreover, A* yields very good results when guided by the great circle distance potential. We attribute this to the fact that the Airway Network was designed so that routes can closely follow the geodesic between origin and destination, which renders A*'s GCD potential highly effective.

We also want to point the reader to the stark difference in preprocessing time. CHs

needs more than 20 minutes preprocessing time (on a single core). While this is still within reasonable bounds, note that A* does not need any preprocessing at all: since it uses the GCD potential, all values for the potential can be computed on the fly. This, on the other hand, also contributes significantly to its running time, making up for more than half of it. This also explains why the static A* is slower than its time dependent counterpart. One could optimise here by precomputing and storing the values for each node.

5.3 Results in the PWL Case

We start analysing the time-dependent case by considering piecewise linear travel time functions. As before, we consider the flight levels 290, 340 and 390, but now we additionally use the weather in the months December, February and March. Following the naming above, we shall again denote our instances by I-*FL-Mon-n*, where FL again stands for the first two numbers denoting the considered altitude, Mon stands for the month in which the weather prognosis is given and the last number is the discretisation step (so -1 stands for 1 hour resolution and -3 for 3 hour resolution).

To compare A* to Dijkstra's algorithm and TCHs, we chose two discretisations of the time interval $[t_0, t_r]$. As our weather prognoses are spaced three hours apart, a discretisation step of three hours is a natural choice. In order to obtain greater precision, we also chose a step of one hour.

Just like in the static case, we list average query times in columns four and seven of Table 5.2. These were again obtained by measuring the time it takes to compute shortest paths for all OD pairs and dividing by the number of pairs. As becomes apparent in the table, A* outperforms TCHs, only this time the speedup is significant: while TCHs yield close to no speedup, A* generates an average speedup of $\times 25$ with respect to Dijkstra. Even with respect to TCHs, A*'s speedup is greater than one order of magnitude (around $\times 16.3$).

Contrary to the static case, we have to do some short preprocessing for A*. This consists of two phases: for each arc a and each PWL $\tilde{T}_a$, find the minimum value $\underline{\tilde{T}}_a$, and afterwards, run an all-to-one Dijkstra for each target node on the resulting lower-bound-graph. Still, this can be achieved on 20 cores literally within seconds. In contrast, we direct the reader's view to the preprocessing time column in TCHs in Table 5.2. Notice that here the time is given in minutes, not seconds – times run from $1\frac{1}{2}$ hours to almost 8 hours (I-34-Feb-1), also on 20 cores. As updated weather prognoses are released every six hours, preprocessing takes far too long to be of any use in our application.

However, one must note that A* preprocessing uses the fact that we know in advance which of the airports and waypoints are possible targets, and was in fact designed with that specific knowledge in mind. TCHs, on the other hand, has no such information and acts on much broader terms.

Instance	DIJK$_{PWL}$	TCHs			A^*_{PWL}		
	query (ms)	prep (min)	query (ms)	speedup ×	prep (s)	query (ms)	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

Table 5.2: Comparison of TCHs and A^*_{PWL} for the PWL case

5.4 Results in the Exact Case

As for the exact case, we only compare the runtimes of Dijkstra's algorithm against those of A*, since the implementation of TCHs does not allow for the exact formula. In Table 5.3, we provide the runtimes of the exact versions of both Dijkstra's algorithm and A*. Again, A* needs some preprocessing time ($\approx$ 7s), in which we compute the optimal wind as described in Section 4.2.2 and run the same all-to-one Dijkstras as in the previous case. The speedup in query times is now around $\times$20, with absolute A* query times ranging in single-digit milliseconds.

We also wish to discuss the quality of our PWL approximation. In a setting where exactness is paramount for safety, it is important that PWLs yield correct results, as calculated lengths are equivalent to the fuel an aircraft has to carry. Both underestimation and overestimation can lead to disastrous results, as heavier aircraft burn more fuel than lighter aircraft (thus ending up with less!), while aircraft carrying too little fuel might not have enough reserves to provide for deviations or holding patterns.

For that reason, Table 5.4 contains an error discussion of the PWL results with respect

Instance	DIJK_E	A^*_E		
	query (ms)	prep (s)	query (ms)	speedup ×
I-29-Dec	100.89	7.51	5.80	17.38
I-34-Dec	102.12	7.38	6.13	16.64
I-39-Dec	104.33	7.56	4.47	23.34
I-29-Feb	100.88	7.66	5.49	18.37
I-34-Feb	101.37	7.35	5.68	17.85
I-39-Feb	104.44	7.45	4.16	**25.09**
I-29-Mar	100.07	7.14	4.85	20.60
I-34-Mar	35.72	5.77	1.85	19.25
I-39-Mar	36.18	5.68	1.59	22.66

Table 5.3: Comparison of DIJK_E and A^*_E

to the exact ones. We again used the discretisation steps one hour and three hours, but also considered a step size of 10 minutes, which is denoted by $\text{A}^*_{\text{PWL}_{10}}$. Thus, we can discuss the influence of the discretisation step on the approximation error.

The error was measured by outputting a path for each search algorithm and recomputing the exact costs on it, thus yielding an optimal path for A^*_E and a possibly suboptimal result for A^*_PWL. While the average error in all approximation cases is very small, there are routes which yield a considerable error (compare the rows I-29-Dec and I-34-Dec, especially for one- and three-hour discretisation). As already discussed in the Introduction (Section 1.1), in aviation, even savings of seemingly low 0.5% can have a noticeable impact on both the environment as well as the financial situation of an airline. This does justify longer running times, especially since errors in the single-digit percentage range can lead to the aforementioned consequences. We therefore also include the number of these **bad paths**, which is the number of paths whose costs deviate by more than 0.5% from the cost of the optimum path.

As one should expect, the number of bad paths decreases when refining the discretisation step from three hours to one hour, and drops significantly when using a step of 10 minutes. Yet, one should keep in mind that with decreasing step size, both preprocessing time and space consumption increase, and that in this thesis, we only considered one layer. If one were to extend the PWL approach to all layers, one would have to scale the preprocessing times in Table 5.4 by the number of flight levels available as well as include all arcs connecting the different layers.

The first data column of Table 5.4 contains the preprocessing times needed to obtain the PWLs. This preprocessing is necessary, because within the framework of our application, it is natural to use the exact version of the TTFs and the piecewise linear functions are not a priori known. This preprocessing time needs to be considered in

Instance	prep (s)	av err (%)	max err (%)	bad paths (#)	
I-29-Dec	46.69	0.078	8.76	740	$A^*_{PWL_3}$
I-34-Dec	47.88	0.093	10.92	940	
I-39-Dec	47.93	0.021	2.65	94	
I-29-Feb	47.03	0.035	5.38	269	
I-34-Feb	48.43	0.049	4.63	431	
I-39-Feb	48.45	0.019	3.60	75	
I-29-Mar	31.26	0.030	5.41	183	
I-34-Mar	32.34	0.022	4.60	111	
I-39-Mar	33.01	0.017	4.74	93	
I-29-Dec	139.41	0.059	8.76	506	$A^*_{PWL_1}$
I-34-Dec	140.81	0.072	5.30	701	
I-39-Dec	140.98	0.018	2.65	79	
I-29-Feb	139.74	0.028	5.38	195	
I-34-Feb	141.32	0.038	4.64	317	
I-39-Feb	140.72	0.015	3.60	51	
I-29-Mar	91.38	0.022	5.37	96	
I-34-Mar	92.78	0.019	4.60	87	
I-39-Mar	95.21	0.016	4.74	89	
I-29-Dec	809.02	0.0039	0.23	0	$A^*_{PWL_{10}}$
I-34-Dec	809.73	0.0039	0.75	1	
I-39-Dec	809.93	0.0037	1.07	1	
I-29-Feb	810.78	0.0039	0.23	0	
I-34-Feb	811.79	0.0038	0.29	0	
I-39-Feb	811.32	0.004	0.29	0	
I-29-Mar	540.3	0.0041	1.10	1	
I-34-Mar	504.84	0.0043	0.26	0	
I-39-Mar	505.67	0.0045	0.29	0	

Table 5.4: The error rating of the PWL version of A*, for three discretisation steps

addition to the 'usual' preprocessing where we run all-to-one Dijkstras, nevertheless we did not consider it in the PWL case, since the piecewise linear TTFs are needed for all of the three algorithms.

When comparing to the exact results, however, another issue comes to light: a closer look at Table 5.3 yields that the maximum runtime for one A* run in the exact case is 6.13ms. Since we have less than 19 000 OD pairs, this yields a total runtime for all OD pairs of less than 116 seconds. Note that even in the case of a one-hour-discretisation, two thirds of the preprocessing times alone are longer than two minutes, not including

the actual query times. In the case of the less error-prone 10-minute-discretisation, the preprocessing times exceed the total runtime of A^*_E on all OD pairs by a factor of more than six. This observation together with possible errors for coarser discretisations renders the PWL approach unsuitable in practice.

6 Conclusion

In this thesis, we considered three different versions of the Horizontal Flight Trajectory Optimisation Problem: one assuming static costs, one assuming piecewise linear travel time functions, and one considering the exact travel times. The first can be formulated as a Shortest Path Problem, while the latter two can be modelled as a Time-Dependent Shortest Path Problem.

These problems have been thoroughly discussed in the literature, particularly in the context of their application in road networks. However, we showed that the advantages of some shortest path algorithms over others do not extend to the Airway Network. Notably, the A* algorithm shows a considerable speedup over both Dijkstra's algorithm and Time-dependent Contraction Hierarchies.

In the static case, we showed that A* and (static) Contraction Hierarchies both yield competitive speedups over Dijkstra's algorithm. Yet, as opposed to CHs, A* needs no preprocessing, but uses the great circle distance as potential function, which can be computed on the fly.

Since we modelled HFTOP as a time-dependent shortest path problem, for both the PWL and the exact case, we established criteria by which to check whether any given instance satisfies the FIFO property, thus guaranteeing that all of the algorithms we considered yield optimal results.

For the PWL case, we also compared Dijkstra's algorithm to A* and TCHs, showing that A* outperforms TCHs by one order of magnitude (a factor of ≈ 16) and yields a speedup factor of more than 25 over Dijkstra's algorithm. At the same time, A* maintains much lower preprocessing times than TCHs: A* turned out to require just seconds of preprocessing, whereas TCHs needed almost 8 hours in extreme cases. This renders solving HFTOP through Contraction Hierarchies unsuitable in practice, as weather prognoses are updated every 6 hours. Whether TCHs can be adapted to the special structure of the Airway Network to achieve better speedups, remains an open question.

For the exact case, we introduced the notion of Super-Optimal Wind as a means to underestimate travel times on arcs. We showed that Super-Optimal Wind yields a feasible potential function for the A* algorithm and assessed its quality through both theoretical bounds and practical application.

Furthermore, we computationally assessed the error of the PWL approximation to the exact case. In the ensuing discussion, we listed reasons as to why we discourage their practical use.

Bibliography

[ADGW12] 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*, chapter Hierarchical Hub Labelling for Shortest Paths, pages 24–35. Springer, Berlin, Heidelberg, 2012.

[Air16] Air Transport Action Group (ATAG). Facts and Figures. Online, `http://www.atag.org/facts-and-figures.html`, 2016. Accessed: 11 July 2016.

[BBH+16] 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. *Proceedings of ATMOS 2016, to appear*, 2016.

[BDG+15] Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. Route Planning in Transportation Networks. Technical report, Microsoft Research, 2015. Updated version of the technical report MSR-TR-2014-4.

[BDS+08] 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*, chapter , pages 303–318. Springer, Berlin, Heidelberg, 2008.

[BDSV09] Gernot Veit Batz, Daniel Delling, Peter Sanders, and Christian Vetter. Time-Dependent Contraction Hierarchies. *Proceedings of ALENEX 2009, SIAM*, 2009.

[BGS08] G. Veit Batz, Robert Geisberger, and Peter Sanders. Time Dependent Contraction Hierarchies – Basic Algorithmic Ideas. Technical report, Karlsruhe Institute of Technology, 2008.

[BOSS13] Pierre Bonami, Alberto Olivares, Manuel Soler, and Ernesto Staffetti. Multiphase Mixed-Integer Optimal Control Approach to Aircraft Trajectory Optimisation. *Journal of Guidance, Control, and Dynamics*, pages 36(5):1267–1277, 2013.

[CH66] Kenneth L. Cooke and Eric Halsey. The Shortest Route Through a Network with Time-Dependent Internodal Transit Times. *Journal of Mathematical Analysis and Applications*, pages 493–498, 1966.

[Cri15] Rob Crilly. Jet stream blasts BA plane across Atlantic in record time. Online, `http://www.telegraph.co.uk/news/worldnews/northamerica/usa/11337617/Jet-stream-blasts-BA-plane-across-Atlantic-in-record-time.html`, 2015. Accessed: 04 July 2016.

[Del08] Daniel Delling. *Proceedings of ESA 2008*, chapter Time-Dependent SHARC Routing, pages 332–343. Springer, Berlin, Heidelberg, 2008.

[DGPW14] Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck. Robust Exact Distance Queries on Massive Networks. Technical report, Microsoft Research, 2014. MSR-TR-2014-12.

[DH08] Peter Deuflhard and Andreas Hohmann. *Numerische Mathematik I. Eine algorithmisch orientierte Einführung*. de Gruyter, 2008.

[Die10] Reinhard Diestel. *Graph Theory*. Springer, Heidelberg, 2010.

[Dij59] Edsger W. Dijkstra. *Numerische Mathematik*, chapter A Note on Two Problems in Connexion with Graphs, pages 269–271. Springer, Berlin, Heidelberg, 1959.

[dJ74] H. M. de Jong. Optimal Track Selection and 3-Dimensional Flight Planning. Technical report, Koninklijk Nederlands Meteorologisch Instituut, 1974.

[Dre69] S. E. Dreyfus. An Appraisal of Some Shortest Path Algorithms. *Operations Research*, 1969.

[DSSW09] Daniel Delling, Peter Sanders, Dominik Schultes, and Dorothea Wagner. *Algorithmics of Large and Complex Networks: Design, Analysis, and Simulation*, chapter Engineering Route Planning Algorithms, pages 117–139. Springer, Berlin, Heidelberg, 2009.

[DW09] Daniel Delling and Dorothea Wagner. *Robust and Online Large-Scale Optimization*, chapter Time-Dependent Route Planning. Springer, Berlin, Heidelberg, 2009.

[Eur16] Eurocontrol. Route Availability Document. Online, `https://www.nm.eurocontrol.int/RAD/`, 2016. Accessed: 06 July 2016.

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

[GSSV12] Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. Exact Routing in Large Road Networks Using Contraction Hierarchies. *Transportion Science*, pages 1–17, 2012.

[HNR68] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, pages 100–107, 1968.

[HSS14] Peter Hecker, Per Martin Schachtebeck, and Meiko Steen. *Handbuch der Luftfahrzeugtechnik*, chapter Flugführung, pages 642–699. Carl Hanser Verlag, 2014.

[Int16a] International Air Transport Association. IATA Price Analysis. Online, `http://www.iata.org/publications/economics/fuel-monitor/Pages/price-analysis.aspx`, 2016. Accessed: 20 July 2016.

[Int16b] International Civil Aviation Organization. Air transport, passengers carried. Online, `http://data.worldbank.org/indicator/IS.AIR.PSGR`, 2016. Accessed, 27 July 2016.

[Kar16] Karlsruhe Institute of Technology. Fast and Exact Route Planning. Online, `http://algo2.iti.kit.edu/routeplanning.php`, 2016. Accessed: 11 July 2016.

[KASS12] Stefan E. Karisch, Stephen S. Altus, Goran Stojković, and Mirela Stojković. *Quantitative Problem Solving Methods in the Airline Industry*, chapter Chapter 6 – Operations, pages 283–383. Springer, Berlin, Heidelberg, 2012.

[KN12] Sven Krumke and Hartmut Noltemeier. *Graphentheoretische Konzepte und Algorithmen*. Springer, Berlin, Heidelberg, 2012.

[KS93] David E. Kaufmann and Robert L. Smith. Fastest Paths in Time-Dependent Networks for Intelligent Vehicle-Highway Systems Application. *IVHS Journal*, pages 1–11, 1993.

[Luf15] Lufthansa Group. Balance – Nachhaltigkeitsbericht der Lufthansa Group 2014. Online, `https://www.lufthansagroup.com/fileadmin/downloads/de/verantwortung/balance-2015-epaper/`, 2015. Accessed: 11 July 2016.

[MdlC16] Pedro Maristany de las Casas. Cost-Minimal Aircraft Trajectories. Master's thesis, Technische Universität Berlin, 2016. To appear.

[OR90] Ariel Orda and Raphael Rom. Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. *Journal of the ACM*, pages 607–625, 1990.