A Priori Search Space Pruning in the Flight Planning Problem

Adam Schienle
Zuse Institute Berlin, Berlin, Germany
schienle@zib.de

Pedro Maristany
Zuse Institute Berlin, Berlin, Germany
maristany@zib.de

Marco Blanco
Lufthansa Systems, Raunheim, Germany
marco.blanco-sandoval@lhsystems.com

Abstract

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

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

2012 ACM Subject Classification Mathematics of computing → Combinatorial optimization

Keywords and phrases time-dependent shortest path problem, crossing costs, flight trajectory optimization, preprocessing, search space

Digital Object Identifier 10.4230/OASIcs.ATMOS.2019.8

1 Introduction

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

Airlines plan a flight by computing an optimal route according to their preferences, which may include minimum fuel, minimum time, minimum cost (e.g., in USD), or a combination thereof. Furthermore, one has to consider various overflight charges for countries, weight dependent fuel consumption functions and weather-dependent arc lengths. Due to the weather being time-dependent, this introduces an implicit time-dependency into the problem. On top, some air navigation service providers publish restrictions to prevent congestion, such as EUROCONTROL's *Route Availability Document*[12].

The **Flight Planning Problem** as we will discuss it is the problem of finding a minimum cost (in USD) trajectory given an origin and a destination airport, a departure time and a weather forecast, an aircraft and its consumption functions, and overflight charges for each country. Note that we take the airline's point of view, and only consider one aircraft at a time. A general introduction to this problem can for instance be found in [18]. We will in the course of this paper disregard restrictions imposed by air navigation service providers (ANSPs), as they can usually only be enforced on a given route. In particular, the Flight Planning Problem as we see it is a base problem which may have to be solved a number of times in order to obtain a valid flight plan.

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

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

1.1 Literature and Related Work

Reliant on a directed graph, the Flight Planning Problem shares similarities with the (Time-Dependent) Shortest Path Problem on road networks. The best-known algorithm to solve the non time-dependent version is Dijkstra's Algorithm[8], which can be extended for the time-dependent setting, see [10]. However, the time-dependent version of Dijkstra's Algorithm is only guaranteed to find an optimal solution if the FIFO property is satisfied; we say that a function $f\colon A \times \mathbb{R}_0^+$ satisfies the FIFO property, if

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

This property basically states that overtaking is not possible by waiting at nodes. Many speedup techniques for Dijkstra's Algorithm exist for both the static and the time-dependent version; [1] provides a good overview. While some such as A*[16] use potential functions to guide the query at runtime, the methods which are most effective on road networks require one or more preprocessing step. Contraction Hierarchies [15] (CHs) and its time-dependent sibling Time-Dependent Contraction Hierarchies [2] (TCHs) assign ranks to the nodes, and look for a shortest path through an in- and then decreasing sequence of nodes.

In [6], the authors show that TCHs do not perform as effectively on the airway network graph as on road networks, being dominated both in preprocessing and in query times by A*. The authors also introduce a novel technique for underestimating traversal times in the flight planning problem. However, they do not consider overflight charges, which are a central component of this paper.

Jensen et al. introduce a geometric algorithm to solve the Free Flight Problem [17]. They partition any free flight airspace into rectangles of equal, constant wind. While similar to Dijkstra's algorithm, their method sorts nodes based not on their costs, but on the costs of their cheapest successor, and lets nodes compete among each other for their successors, thus achieving a speedup over Dijkstra's Algorithm or A*. One has to note, however, that their approach assumes airspace users can choose their waypoints freely. On the other hand, we will assume that all waypoints and segments are defined, and that one is not allowed to fly via self-defined waypoints. This represents the current point of view as expressed by air navigation service providers such as EUROCONTROL [11].

In terms of overflight charges or crossing costs, relatively little research has been conducted. Most notably, Blanco et al. define and analyse the Shortest Path Problem with Crossing Costs in [7], and solve a special case to optimality. In [5], the authors furthermore propose a cost projection onto the arcs. Dreves and Gerdts [9] give an example on how to solve the problem using optimal control, albeit in a bounded region in Europe. None of the works cited deal with overflight charges considers the influence of weather. Both [6] and [5] only run queries on one layer of the airway network, whereas we use the full (3D) graph for our computations.

We seek to cut nodes with several different methods from the a-priori search space, pruning all those nodes which provably cannot lie on an optimal path. A similar technique is used in other applications of shortest path problems, e.g. in Electric Vehicle Routing [4], or in the algorithm Approximated Time-Dependent Contraction Hierarchies [3].

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

In section 2, we develop the theory on how to prune nodes prior to the query. Section 3 shows how to model the Free Flight Problem as a Time-Dependent Shortest Path Problem with Crossing Costs. We develop several different pruning methods for the Free Flight Problem in section 4, and show the results of real-world test cases in section 5.

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

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

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

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

In [6], the authors show that in flight planning, A* outperforms even the most promising speedup technique to Dijkstra's Algorithm, Time-Dependent Contraction Hierarchies[2]. Since our pruning techniques in section 4 are similar to A*, we limit ourselves to the discussion of pruning the search space for Dijkstra's algorithm.

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

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

▶ **Theorem 1.** *Assume we are given a* TDSPP *as defined above, and an upper bound* $\overline{c}$ *on the costs for a shortest path between* s *and* t*. Let* $\underline{c}_s^t \colon V \to \mathbb{R}_0^+$ *be a function which underestimates the costs for a shortest* s*-*t*-path via* v*. Any node* $v \in V$ *which violates the pruning inequality*

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

cannot lie on an optimal path.

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

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

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

▶ Remark 2. In practice, we obtain a function $\underline{c}_s^t(v)$ as required in the above theorem by computing a lower bound function $\underline{c} \colon A \to \mathbb{R}_0^+$ such that

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

This approach is common for shortest path problems (e.g. for A* or ALT, see [1]), and is also used in [6]. We write $\underline{G} := (G, \underline{c})$, and run a one-to-all Dijkstra from s and an all-to-one Dijkstra to t on $\underline{G}$. Note that $\underline{G}$ carries static arc costs, hence we can grow the Dijkstra trees in $\mathcal{O}(|A| + |V| \log |V|)$.

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

Theorem 1 states that any path whose lower-bound costs are already higher than a pre-computed upper bound cannot be optimal. Note that this mimics the A* algorithm[16]. The key difference is that by using Theorem 1, nodes which cannot lie on an optimal path are eliminated a priori, rather than being discarded during the search. This means that it is a little weaker than A*: for any node $v \in V$, the latter adds an estimate of the remaining costs from v to t to the actual costs $c(P_s^v)$ from s to v, and sorts the node in the heap accordingly. Contrastingly, when applying Theorem 1, one adds two cost estimates and compares them to an upper bound. Unless the estimate is perfect, this leads to a gap between $\underline{c}(P_s^v)$ and $c(P_s^v)$. However, in practice it might be easier to compute an underestimation for the costs of a complete path, rather than just parts of it. Also, a pre-pruned search space can be advantageous for search algorithms which do not maintain a heap structure; e.g., by sorting the graph's nodes in topological order and run a search algorithm exploring the resulting node groups in their respective order.

▶ Remark 3. The applicability of Theorem 1 is dependent on both the quality of the lower bound as well as the quality of the upper bound solution. Clearly, lowering the upper bound will result in more nodes being eliminated, as will raising the lower bound.

As shown by Fredman et al.[14], the runtime for the static Dijkstra's Algorithm is $\mathcal{O}(|A| + |V| \log |V|)$ when a Fibonacci heap is used to store the unprocessed labels. The dynamic, i.e., time-dependent case with FIFO travel time functions can be solved using almost the same version of the algorithm [10]. The only difference is the evaluation of the arcs' cost: in the static case it can be in constant time but in the dynamic case the complexity of the evaluation depends on the shape of the functions [13]. This is why, even in the FIFO case, it is not guaranteed that the TDSPP is polynomially solvable and this is also the motivation for a restriction of the search space before the query commences.

3 Modelling the Free Flight Problem

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

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

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

Fuel costs depend linearly on how much fuel an aircraft burns en route. The fuel burn is directly proportional to the distance relative to the surrounding air mass, or *air distance*. As in [17, 6], we assume that an aircraft flies with constant speed, which in turn renders the air distance proportional to the time en route. Therefore, we can again think of fuel costs as being charged per arc, and write

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

Both fuel and time costs naturally depend on the travel time, which in turn is weather-dependent. We use the same travel time functions as given [6], assuming that weather is given for a discrete point set $\Delta \subset \mathbb{R}_0^+$ in time over a long enough interval to cover all flight durations. Usually, the break points are three hours apart; for any $\tau \notin \Delta$, we interpolate the closest two weather data objects to obtain a wind vector $\mathbf{w}(a, \tau)$ for an arc $a \in A$. It can be decomposed into its cross wind component $w_c(a, \tau)$ perpendicular to the direction of flight, and a track wind component $w_t(a, \tau)$ parallel to it. Together with the constant air speed, this leads to the travel time (c.f. [6, 17])

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

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

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

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

For the definition of crossing costs for regions, we will closely follow the notation presented by Blanco et al. in [7]. We write $\delta^+(v)$ for out-arcs and $\delta^-(v)$ for in-arcs of the node v, and define $\delta(v) := \delta^+(v) \sqcup \delta^-(v)$. We assume that the arcs are partitioned into a set $\mathcal{R}$ of k regions

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

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

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

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

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

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

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

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

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

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

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

▶ Remark 4. T-SPROC is an extension of TDSPP. Clearly, the first sum constitutes the time-dependency, whereas the second one accounts for the crossing costs. Especially, if $c_o^R(\cdot) \equiv 0$, we re-obtain the TDSPP. Also, note that while the crossing costs c_o are not time-dependent, they are not defined per arc. This obvious difference to TDSPP poses an additional challenge.

We have shown in [19] that under certain conditions, the wind functions in our application satisfy the FIFO property; we will for the remainder of this paper presuppose the FIFO property for c^A. Blanco et al. developed the Two-Layer-Dijkstra algorithm in [7], which solves the shortest path problem with crossing costs to optimality in polynomial time.

We also observe that Theorem 1 requires an underestimation function for all nodes $v \in V$. Since we cannot even evaluate the crossing costs for non-boundary nodes $v \in V \setminus \mathcal{B}$, we cannot apply Theorem 1 to T-SPROC directly. However, since overflight charges are always non-negative, we can underestimate the function c_o by zero.

4 Preprocessing in Practice

In order to have as low runtimes as possible, we aim to prune the a-priori search space for the Flight Planning Problem. Regardless of which pruning algorithm we choose, the objective function will always be given by (1).

4.1 Dead-End elimination

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

4.2 The Tank-Capacity Pruning

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

To underestimate $\varphi_s^t(v)$ for a node $v \in V$, we use the Super-Optimal Wind as introduced by Blanco et al. in [6]. The Super-Optimal Wind for an arc $a \in A$ is an artificial wind vector which never underestimates the travel time for an arc $a \in A$ arising out of an actual wind conditions. Given $a \in A$, it is obtained by separately minimising the cross wind and maximising the track wind, leading to the artificial wind vector

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

where $\underline{w}_c(a) := \min_{\tau \in \mathbb{R}_0^+} |w_c(a, \tau)|$ and $\overline{w}_t(a) := \max_{\tau \in \mathbb{R}_0^+} w_t(a, \tau)$. Note that one can obtain better lower bounds by computing the Super-Optimal Wind $\mathbf{w}_i(a)$ per $\tau_i, \tau_{i+1} \in \Delta$. For a full discussion of the topic, we point the reader to [6]. As stated in the same work, Super-Optimal Wind can be pre-computed independent of the instance in a few seconds for all arcs $a \in A$.

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

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

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

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

where $\underline{l}(u, w)$ denotes the length of a shortest path from u to w in G, with respect to $\underline{l}$. This value is a lower bound on the wind-corrected distance between s and t. We convert $\underline{l}_s^t(v)$ to fuel consumption by assuming an optimal flight profile on the given air distance, to obtain a lower bound $\underline{\varphi}_s^t(v)$ on the fuel consumption via v. Whenever $\underline{\varphi}_s^t(v)$ exceeds the tank capacity $\overline{\Phi}$, we can eliminate v from the search space. Note that while this is similar to Theorem 1, we do not rely on a precomputed upper bound; rather, the tank capacity is implicit in the input data. Since all arc costs are static, we can compute this lower bound for every node in $\mathcal{O}(|A| + |V| \log |V|)$.

4.3 Fuel and Time Pruning

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

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

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

Hence, given an upper bound solution with cost $\overline{c}$, the pruning inequality given in Theorem 1 evaluates to

$$\underline{c}_f(v) + \underline{c}_t(v) =: \underline{c}^A(v) \leq \overline{c}, \tag{2}$$

and we can eliminate any node $v \in V$ which violates it. This is essentially the application of Theorem 1 to the time-dependent part of T-SPROC, with crossing costs underestimated by the constant zero function. As in the tank capacity case, all data is static. In particular, we can compute this lower bound at the cost of Dijkstras, namely in $\mathcal{O}(|A| + |V| \log |V|)$.

4.4 Pruning Crossing Costs

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

Overflight charges were already investigated in [5], where the authors project crossing costs for regions on the arcs using a heuristic which works very well in practice, but cannot guarantee an underestimation. Instead, we are going to pursue an exact solution. In [7], Blanco et al. introduce a macro graph which they use to keep track of the overflight costs. We will mimic this construction, and use the macro graph to underestimate the overflight costs.

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

Since crossing costs are not time-dependent, we can precompute a lower bound on them by constructing a new graph $M = (V', A')$. We define this **macro graph** M as in [7]:

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

i.e., the new nodes are all boundary nodes of the regions. We also count airport nodes as boundary nodes, to allow for crossing costs for flights beginning or ending in the interior of a region. While instead of all airports it would suffice to only add s and t (as done in [7]), our more general definition has the advantage of being independent of the instance. We can hence reuse the same graph structure for all flight instances.[1] We then set

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

In other words, we insert an arc between two boundary nodes of R whenever there is a path connecting them which is entirely contained in R. We then endow M with the (not time-dependent) metric function d as defined in section 3. The macro graph for a set of airspaces is depicted in Figure 1.

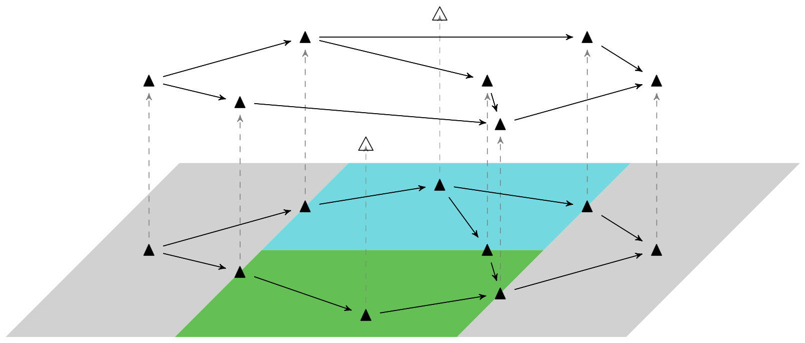

Figure 1 Macro graph with two airports in the gray regions.

Note that for all boundary nodes $b \in \mathcal{B}$, the value

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

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

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

[1] While our definition is very similar to the one in [7], the authors use it to solve the shortest path problem with crossing costs to optimality, whereas we use it to prune the search space.

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

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

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

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

4.5 Pruning in Practice

The airway network is designed as a directed graph on the Earth's surface and the flight levels can be thought of as distinct, interconnected layers of this graph. To quicken the preprocessing step, we only consider the base layer of the airway network: we set an arc's underestimated length to the minimum lower bound length over all flight levels on which the arc is defined. This may weaken the effect of the pruning algorithms, but does not affect correctness. We use this procedure in our computations.

5 Computational Results

Both the Airway Network and the instance data were provided by Lufthansa Systems. The graph consists of 109 314 nodes and 838 114 arcs per level, on 43 such flight levels. The instance set consists of the 7 735 most often flown international connections, based on week 21 of 2014.[2] We limit ourselves to international relations only, since for some countries such as the US, overflight charges do not apply for domestic flights.

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

We implemented all algorithms explained in Section 4 in `C++` within our flight planning tool. We compiled the code with `GCC`, and all our tests were carried out on machines with 132GB of RAM, and an Intel(R) Xeon(R) CPU E5-2690 v4 processor with 2.60GHz and 35.8MB of cache. Queries were run in single-thread mode.

We will measure the quality of the pruning methods by comparing runtimes and the number of active nodes before pruning to afterwards both absolutely and relative to dead-end elimination, which will act as our baseline. A higher number always indicates a more effective method. The advantage of counting active nodes is that the speedup is purely algorithm- but not implementation-dependent. For a fixed instance, we will keep the same upper bound solution for each underestimator for all pruning algorithms. The names for the different pruning methods found in this section are listed in Table 1.

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

Table 1 Pruning methods.

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

We investigate three different scenarios: we use two different weather prognoses, six months apart (`wth-16` and `wth-17`), and in a third case investigate the situation where there is no wind at all. This `gcd` case highlights the potential for FUEL TIME OFC pruning while eliminating unpredictable effects introduced by the wind. The absolute runtimes for each scenario, averaged over all 7 735 connections, are presented in Table 2.

Table 2 Average Query Times per Weather.

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

Recall that we investigate the base problem in flight planning, that of finding a 3D trajectory. Note, too, that we may have to recompute a solution given new restrictions imposed by ANSPs. With this in mind, it is imperative that each instance can be computed as fast as possible. To this end, we compare the query time of Dijkstra's Algorithm after applying TANK CAP, FUEL TIME, and FUEL TIME OFC pruning to the query time after using only DEAD END. For each of the more than 7 000 flights, the speedup is recorded both absolutely and relatively, averaged over the instances, and then summarised for the current weather situation. Note that this approach is possible since our runtimes are in the order of seconds rather than milliseconds, which yields comparatively stable runtime measurements. The previous table already indicates the results, but we also visualise them in Figure 2.

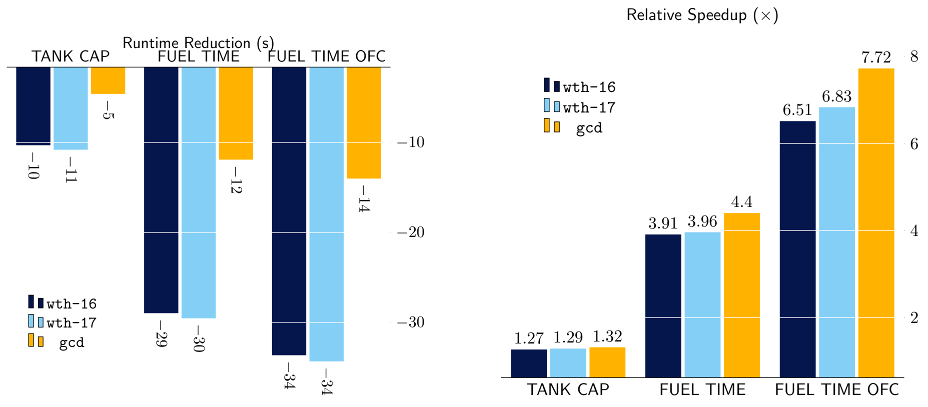

Figure 2 Absolute runtime reduction (left) and averaged relative speedup per instance (right), visualised per test set.

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

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

Table 3 Average 2D Search Space Reduction Per Instance.

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

As one would expect, FUEL TIME OFC pruning is the most effective method. Applying FUEL TIME OFC pruning to the search space yields a reduction of more than 90% of the active nodes. This reduction is also visible in terms of the average query time speedup, which is even ≈ 1.72 times better than with the second best method, FUEL TIME pruning. While the TANK CAP pruning is not quite as effective as the other methods, its inherent advantage is that it does not rely on an upper bound solution. Indeed, the upper bound is given by the input in terms of the aircraft's tank capacity, which renders it a computationally light alternative – or a fallback method in case one cannot find a reasonable upper bound solution.

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

6 Conclusion

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

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

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

References

1 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.

2 G. Veit Batz, Robert Geisberger, Peter Sanders, and Christian Vetter. Minimum Time-dependent Travel Times with Contraction Hierarchies. *J. Exp. Algorithmics*, 18:1.4:1.1–1.4:1.43, April 2013.

3 Gernot Veit Batz, Robert Geisberger, Sabine Neubauer, and Peter Sanders. Time-Dependent Contraction Hierarchies and Approximation. In Paola Festa, editor, *Experimental Algorithms*, pages 166–177, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

4 Moritz Baum, Julian Dibbelt, Lorenz Hübschle-Schneider, Thomas Pajor, and Dorothea Wagner. Speed-Consumption Tradeoff for Electric Vehicle Route Planning. In Stefan Funke and Matúš Mihalák, editors, *14th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*, volume 42 of *OpenAccess Series in Informatics (OASIcs)*, pages 138–151, Dagstuhl, Germany, 2014. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/OASIcs.ATMOS.2014.138.

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

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

7 Marco Blanco, Ralf Borndörfer, Nam Dung Hoang, Anton Kaier, Thomas Schlechte, and Swen Schlobach. The Shortest Path Problem with Crossing Costs. Technical report, ZIB, 2016.

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

9 Axel Dreves and Matthias Gerdts. Free Flight Trajectory Optimization and Generalized Nash Equilibria in Conflicting Situations. Technical report, Universität der Bundeswehr München, 2017.

10 Stuart E. Dreyfus. An Appraisal of Some Shortest Path Algorithm. *Operational Research*, 17(3):395–412, June 1969.

11 Eurocontrol. Free route airspace (FRA). Online, 2019. Accessed April 2019. URL: https://www.eurocontrol.int/articles/free-route-airspace.

12 Eurocontrol. Route Availability Document. Online, April 2019. URL: https://www.nm.eurocontrol.int/RAD/.

13 Luca Foschini, John Hershberger, and Subhash Suri. On the Complexity of Time-Dependent Shortest Paths. *Algorithmica*, 68(4):1075–1097, April 2014. doi:10.1007/s00453-012-9714-7.

14 Michael L. Fredman and Robert Endre Tarjan. Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms. *Journal of the ACM*, 34(3):596–615, July 1987.

15 Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. Exact Routing in Large Road Networks Using Contraction Hierarchies. *Transportation Science*, 46(3):388–404, August 2012.

16 Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions of Systems Science and Cybernetics*, 4(2):100–107, July 1968.

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

18 Stefan E. Karisch, Stephen S. Altus, Goran Stojković, and Mirela Stojković. *Quantitative Problem Solving Methods in the Airline Industry*, chapter Operations, pages 283–383. Springer, 2012.

19 Adam Schienle. Shortest Paths on Airway Networks. Master's thesis, Freie Universität Berlin, 2016.