Heuristic Variants of A* Search for 3D Flight Planning

Anders N. Knudsen, Marco Chiarandini, and Kim S. Larsen(✉)

Department of Mathematics and Computer Science,
University of Southern Denmark, Campusvej 55, 5230 Odense M, Denmark
{andersnk,marco,kslarsen}@imada.sdu.dk

Abstract. A crucial component of a flight plan to be submitted for approval to a control authority in the pre-flight phase is the prescription of a sequence of airways and airway points in the sky that an aircraft has to follow to cover a given route. The generation of such a path in the 3D network that models the airways must respect a number of constraints. They generally state that if a set of points or airways is visited then another set of points or airways must be avoided or visited. Paths are then selected on the basis of cost considerations. The cost of traversing an airway depends, directly, on fuel consumption and on traversing time, and, indirectly, on weight and on weather conditions.

Path finding algorithms based on A* search are commonly used in automatic planning. However, the constraints and the dependency structure of the costs invalidate the classic domination criterion in these algorithms. A common approach to tackle the increased computational effort is to decompose the problem heuristically into a sequence of horizontal and vertical route optimizations. Using techniques recently designed for the simplified 2D context, we address the 3D problem directly. We compare the direct approach with the decomposition approach. We enhance both approaches with ad hoc heuristics that exploit the expected appeal of routes to speed-up the solution process. We show that, on data resembling those arising in the context of European airspaces, the direct approach is computationally practical and leads to results of better quality than the decomposition approach.

1 Introduction

The Flight Planning Problem (FPP) aims at finding, for a given aircraft, a sky trajectory and an initial fuel load, minimizing the total cost determined by fuel consumption and travel time. The sky is subdivided into airspaces where airway points and airways between them are predefined. Thus, a route, denoted by origin and destination airports, corresponds to a path in a network that models the 3D space. The path starts at the origin airport, climbs to a favorable altitude and finally descends to the destination airport while satisfying different types

K. S. Larsen—Supported in part by the Independent Research Fund Denmark, Natural Sciences, grant DFF-7014-00041.

© Springer International Publishing AG, part of Springer Nature 2018
W.-J. van Hoeve (Ed.): CPAIOR 2018, LNCS 10848, pp. 361–376, 2018.
https://doi.org/10.1007/978-3-319-93031-2_26

of constraints. There are clear operational, financial, and environmental motivations for aiming at feasible and cost-optimal paths. However, the size of airway networks, the nature of a large number of the constraints, and the dependencies among different factors that affect costs make the problem substantially more difficult to solve compared with classic shortest path problems in road networks. In addition, some of the constraints from central control authorities (Eurocontrol in Europe or FAA in USA) are issued to take rapidly changing traffic conditions into account. Thus, they are updated frequently, and airlines, and to an even larger extent small private airplanes, tend to determine the precise flight route only a few hours before take-off, when more constraints are available. Although the route can be adjusted during operation, doing so will lead to a suboptimal route compared with a complete route determined offline. As take-off approaches, available processing time will be on the order of a few seconds.

The cost of flying through an airway depends on the time when the traversal occurs and the fuel consumed so far. There is a direct dependency between the two, because airlines calculate the total cost as a weighted sum of total travel time and total fuel consumption. There is also an indirect dependency, because the time spent and amount of fuel consumed on an arc depend on the performance of the aircraft, which is influenced by: (i) the weather conditions, which in turn depend on the time when the arc is traversed, and (ii) the weight, which in turn depends on the fuel consumed so far. A consequence of these dependencies is that the cost of each arc of the airway network is not statically given but becomes known only when the path to a node is determined. A further complication is that while the departure time is fixed, the initial amount of fuel is to be decided. This cost structure leads to issues similar to time dependencies in shortest path on road networks [1].

The indirect cost dependencies together with the impossibility of waiting at the nodes, even at the departure airport, mean that it might be a disadvantage to arrive cheaply at a node (and hence earlier or lighter than other alternatives) since it may preclude obtaining high savings on the remaining path, due to favorable weather conditions developing somewhere at a later time. In other words, the so-called First-In-First-Out (FIFO) property cannot be assumed to hold. For labeling algorithms typically used to approach routing problems, Dijkstra [2] and A* [3], this means that a total ordering of the labels at the nodes is not available, which has a tremendous impact on efficiency.

Further complexity is added by the quantity and type of constraints. The most challenging ones are Route Availability Document (RAD) constraints. They include local constraints affecting the availability of airways and airspaces at certain times, but most of them are conditional constraints. For example, if the route comes from a given airway, then it must continue through another airway. Or some airways can only be used if coming from, or arriving to, certain airspaces. Or flights between some locations are not allowed to cross certain airspaces at certain flight levels. Or short-haul and long-haul flights are segregated in congested zones.

Due to increasing strain on the European airspace, the networks become more heavily constrained to ensure safe flights. There are typically more than 16,000 constraints in the European network and they can be updated several times a day. Some of the RAD constraints are generalizations of *forbidden pairs*, which make the problem at least as hard as the *path avoiding forbidden pairs* problem, shown to be NP-hard [4]. Additionally, RAD constraints invalidate the FIFO assumption, with far-reaching effects in terms of complexity. Therefore, a common approach in industry is to implement a first stage, where the algorithms ignore some constraints that are difficult to formalize and check during route construction, and enforce them later by somehow repairing the route. All constraints are verified by the control authority when a route is submitted for approval.

A common approach in industry is to solve the FPP in a decomposed form: find a horizontal 2D route and then determine a vertical profile for it. While this reduces the computation cost considerably, it does not guarantee optimality. However, due to the practical relevance, recent research has focused on these two simplified problems. In [5], the authors address the horizontal 2D flight planning problem without constraints and study methods for calculating weather-dependent estimates for the heuristic component of A* algorithms. They show that with these estimates, A* achieves better experimental results in both preprocessing and query time than contraction hierarchies [6], a technique from state-of-the-art algorithms for shortest path on road networks. In [7], we proposed a framework for handling the RAD constraints in the horizontal 2D problem. We introduced a tree representation for constraints to maintain them during path construction, and compared lazy techniques for efficiency. Experimental work suggested that the best running times were obtained using an approach similar to a logic-based Benders decomposition with no-good cuts, in which we ignore constraints in the initial search and only add those that are violated in the subsequent searches. In [8], we focused on the impact of the FIFO assumption on the vertical route optimization problem, constrained to a given horizontal route, but not considering constraints. We showed that wrt. cost considerations, heuristically assuming the FIFO property in vertical routing is unlikely to lead to suboptimal solutions.

We extend our work on constraint handling to the 3D context to show that an A* approach can be practical in solving the FPP directly in a 3D network. With respect to our 2D work [7], the networks we consider increase in size from 11,000 nodes and 1,000,000 arcs to 200,000 nodes and 124,000,000 arcs. We design heuristics to improve the efficiency of the algorithms and empirically study the quality loss that they imply. We focus on two types of heuristics: those affecting the estimate of the remaining cost from a node to the goal in the A* framework and those using the desired shape of the vertical profile to prune label expansion in the A* algorithm. We present the comparison of the most relevant among these heuristics and combinations. We include the comparison with the obvious consequent outcome from our previous research, namely a two-phase approach with constraints handled lazily by restart as in [7] and costs as in [8]. Our results

demonstrate that a direct 3D approach leads to better results in terms of quality, with computation times that remain suitable for practical needs.

There are elements from real-life problems we ignore: First, we do not consider the initial fuel as a decision variable. We assume this value is given on the basis of historical data. In practice, we would approach the problem using a line-search method as in [8]. Second, we do not consider (possibly non-linear) overflight costs associated with some airspaces. We refer to [9] for a treatment of these issues; authors suggest a cost projection method to anticipate the cost incurred by the overflight and then eliminate potential paths. Finally, not all airspaces have predefined airway points; free route airspaces lead to algorithms exploiting geometric properties, and they have been shown to be more efficient than graph-based ones [10] in those scenarios. These algorithms could in principle be integrated with the framework presented here, calling them as subroutines when an entry point to a free route airspace is expanded.

This work is in collaboration with an industrial partner. Many of its customers are owners of private planes who plan their flights shortly before departure and expect almost immediate answers to the portable device. Thus, a query should be answered in a few seconds.

2 The Flight Planning Problem

The 3D airspace is represented by *waypoints* that can be traversed at different altitudes (*flight levels*). Waypoints are connected by *airways*. This gives rise to a network in the form of a directed graph $D = (V, A)$. The nodes in V represent waypoints at different latitude, longitude, and altitude. The arcs in A connect all nodes that can be traced back to two different waypoints connected by an airway and whose implied difference in altitude can be operated by the aircraft. Each arc has associated resource consumption and costs. The resource consumption for flying via an arc $a \in A$ is defined by a pair $\boldsymbol{\tau}_a = (\tau_a^x, \tau_a^t) \in \mathbb{R}_+^2$, where the superscripts x and t denote the fuel and time components, respectively. The cost c_a is a function of the resource consumption, i.e., $c_a = f(\boldsymbol{\tau}_a)$.[1] A *3D (flying) route* is an (s, g)-path in D, represented by n nodes, a departure node (source) s, and an arrival node (goal) g, that is, $P = (s, v_1, \ldots, v_n, g)$, with $s, v_i, g \in V$ for $i = 1...n$, $v_i v_{i+1} \in A$ for $i = 1...n-1$, and $sv_1, v_n g \in A$. The cost of the route P is defined as $c_P = c_{sv_1} + \sum_{i=1...n-1} c_{v_i v_{i+1}} + c_{v_n g}$.

The route must satisfy a set $\mathcal{C}$ of *constraints* imposed on the path. These constraints are of the following type: if a set of nodes or arcs A is visited, then another set of nodes or arcs B must be avoided or visited. The visit or avoidance of the sets A and B can be further specified by restrictions on the order of nodes in the route, on the time window, and on the flight level range.

Definition 1 (Flight Planning Problem (FPP)). *Given a network* $N = (V, A, \boldsymbol{\tau}, \boldsymbol{c})$*, a departure node* s*, an arrival node* g*, and a set of constraints* $\mathcal{C}$*,*

[1] The total cost is calculated as a weighted sum of time and fuel consumed. In our specific case, we have used 1.5$ per gallon of fuel and 1000$ per hour.

find an (s, g)*-path* P *in* $D = (V, A)$ *that satisfies all constraints in* $\mathcal{C}$ *and that minimizes the total cost,* c_P*.*

All constraints can be categorized into two classes: *forbidden* and *mandatory*. Constraints consist of an *antecedent* and a *consequent* expression, p and q. A constraint of the mandatory type is *satisfied* when $p \rightarrow q$ and *violated* otherwise; one of the forbidden type is *satisfied* when $p \rightarrow \neg q$ and *violated* otherwise. The expressions p and q give Boolean values. They contain terms that express possible path choices, such as passing through a node (representing a specific flight level at an airway point), traversing an arc (possibly belonging to an airspace), departing or arriving at a node, or visiting a set of nodes in a given order.

3 A* Search with Constraints

We use A* search as our base algorithm, and the following strategy, detailed and shown effective in [7], to handle the numerous constraints: First, we find a path using A*, ignoring all constraints. Then, the path obtained is checked against all constraints. If no constraint is violated, the path is feasible and the procedure terminates. Otherwise, the violating constraints are added to the input and a new search is initiated. The new path will not violate any of the included constraints but it could violate others. Thus, the procedure is repeated until a feasible path is found. In the iterations that follow the initial one, the search must handle the constraints during the construction. A template for A* modified to handle constraints is given in Algorithm 1.

In Algorithm 1, the function FindPath takes the initial conditions as input: a network $N = (V, A, \boldsymbol{\tau}, \boldsymbol{c})$ built using information from the airspace, aircraft performance data, and weather conditions; the query from s to g; the initial fuel load τ_s^x and the departure time τ_s^t; an array $\boldsymbol{h}$ of values $h(v)$ for every node $v \in V$, indicating the precomputed, static, *estimated cost* of flying from that node v to the destination node g; the set of constraints (see below). The time and fuel consumptions for each arc $\boldsymbol{\tau}$ depend on: (i) the flight level, (ii) the weight, (iii) international standard atmosphere deviation (temperature), (iv) the wind component, and (v) the cost index.[2] Inputs (ii), (iii), and (iv) depend on the time of arrival at the arc and hence on the partial path. We can regard these values as retrieved from data tables but we refer to Sect. 5 for more details.

We represent partial paths under construction in Algorithm 1 by labels. A *label* ℓ is associated with a node $\phi(\ell) = u \in V$ and contains information about a partial path from the departure node $s \in V$ to the node u, that is, $P_\ell = (s, ..., u)$.

Labels also have constraints associated. We represent constraints as trees, where leaves are terms and internal nodes are logical operators (AND, OR, NOT). Constraint trees are, potentially, associated with each label. Initially, all constraint terms are in an unknown state and labels have no constraints associated. Then, if during the extension of a path of a label, a term of a constraint

[2] The cost index is an efficiency ratio between the time-related cost and the fuel cost, decided upon at a strategic level and unchangeable during the planning phase.

```
1  Function FindPath(N = (V, A, τ, c), s, g, (τ_s^x, τ_s^t), h, Γ)
2      initialize the open list Q by inserting ℓ_s = ((s), 0, {})
3      initialize ℓ_r = ((), ∞, {})
4      while Q is not empty do
5          ℓ ← retrieve and remove the cheapest label from Q
6          if (c_ℓ + h(φ(ℓ)) > c_{ℓ_r}) then break                ▷ termination criterion
7          if (φ(ℓ) = g) and (c_ℓ < c_{ℓ_r}) then
8              ℓ_r ← ℓ
9              continue
10         foreach node v such that uv in A do
11             ℓ' ← label at v expanded from ℓ
12             evaluate constraint trees in Δ_{ℓ'}
13             if one or more constraints in Δ_{ℓ'} are violated then
14                 continue
15             Insert(ℓ', Q)
16     return P_{ℓ_r} and c_{ℓ_r}

17 Function Insert(ℓ', Q)
18     foreach label ℓ ∈ Q with φ(ℓ) = φ(ℓ') do
19         if (c_ℓ > c_{ℓ'}) then
20             if (Δ_ℓ is implied by Δ_{ℓ'}) then                ▷ ℓ is dominated
21                 remove ℓ from Q
22         else if (c'_ℓ > c_ℓ) then
23             if (Δ_ℓ is implied by Δ_{ℓ'}) then return          ▷ ℓ' is dominated
24     insert ℓ' in Q
25     return
```

Algorithm 1. An A* search template for solving FPP.

is resolved, the corresponding constraint becomes *active* for that label. After resolution, the term is removed from the constraint tree and the truth value is propagated upwards. The evaluation of the constraint terminates when the root node is resolved. The constraint is satisfied if the root node evaluates to false.

In practice, we translate the set of constraints $\mathcal{C}$ into a dictionary of constraint trees Γ with constraint identifiers as keys and the corresponding trees as values. For a constraint $\gamma \in \Gamma$, we let $\iota(\gamma)$ denote the constraint identifier and $T(\gamma)$ the corresponding tree. Then, for each node, $v \in V$, and each arc, $uv \in A$, we maintain a set of identifiers of the constraints that have those nodes or arcs, respectively, as leaves in the corresponding tree. We denote these sets E_v and E_{uv}, with $E_v = \{\iota(\gamma) \mid \gamma \in \Gamma, v \text{ appears in } \gamma\}$ and E_{uv} defined similarly.

We denote by Δ_ℓ the set of constraint trees copied from Γ to a label ℓ when they became active for ℓ. These constraint trees may be reduced immediately after copying because some terms have been resolved. Let $\rho(\gamma, P_\ell)$ be the tree $T(\gamma)$ after propagation of the terms in P_ℓ. Then, we can formally define, $\Delta_\ell = \{\rho(\gamma, P_\ell) \mid \iota(\gamma) \in E_u \cup E_{uv}, u, v \in P_\ell\}$.

In Algorithm 1, each label ℓ is thus the information record made of $(P_\ell, c_\ell, \Delta_\ell)$. We maintain all labels that are created in a structure $\mathcal{Q}$, called the *open list*. At each iteration of the while loop in Lines 5–15 of FindPath, a label ℓ is selected for extraction from the open list if its evaluation, given by $c(\ell) + h(\phi(\ell))$, is the smallest among the labels in $\mathcal{Q}$. Successively, the label ℓ with $\phi(\ell) = u \in V$ is *expanded* along each arc $uv \in A$, and new labels $\ell' = ((s, \ldots, u, v), c_\ell + c_{uv}, \Delta_{\ell'})$ are created (Lines 10–12).[3] The new set of constraint trees $\Delta_{\ell'}$ is obtained by copying the trees from Δ_ℓ, and the trees from Γ identified by E_v and E_{uv}. While performing these operations, the trees are reduced based on the resolution of the terms u and/or uv. If the root of a constraint tree in $\Delta_{\ell'}$ evaluates to true, then the label ℓ' is deleted, because the corresponding path became infeasible (Lines 13–14). On the other hand, if a root evaluates to false, then the corresponding constraint tree is resolved but is kept in $\Delta_{\ell'}$ to prevent re-evaluating it if, at a later stage, one of the terms that were logically deduced appears in the path. Formally, $\Delta_{\ell'} = \{\rho(\gamma, P_{\ell'}) \mid \gamma \in \Delta_\ell\} \cup \{\rho(\gamma, P_{\ell'}) \mid \iota(\gamma) \in E_{\phi(\ell')} \cup E_{\phi(\ell)\phi(\ell')}\}$.

If a new label ℓ' is not deleted, it is proposed for insertion in $\mathcal{Q}$ to the function Insert. The function Insert takes care of checking the domination between the proposed label ℓ' and the other labels in $\mathcal{Q}$ at the same node. The domination criterion plays a crucial role in the efficiency of the overall algorithm.

The loop in the function FindPath terminates on Line 6 when the goal g has been reached and the incumbent best path to g has total cost less than or equal to the evaluation of all labels in $\mathcal{Q}$. Under some well known conditions for $\boldsymbol{h}$ that we describe in Sect. 4.1, the solution returned is optimal.

Label Domination: We motivate our specific choice for the definition of the domination criterion using the following simple example taken from [7]. Let $C_1 = ((a \vee b) \wedge c)$ be the only constraint present. Let $\ell_1 = ((s, a, x), 3, \{C_1\})$, $\ell_2 = ((s, d, x), 4, \varnothing)$ and $\ell_3 = ((s, b, x), 2, \{C_1\})$ be the only three labels at x. Consider the labels ℓ_3 and ℓ_2. Although ℓ_3 is cheaper than ℓ_2, ℓ_2 has not activated C_1 and hence its path ahead is less constrained than ℓ_3. Indeed, ℓ_2 can use the cheapest route to g, while ℓ_3 must avoid c. It is therefore good not to discard ℓ_2, in spite of being more expensive. The labels ℓ_1 and ℓ_3 are identical with regards to constraints. However, the cost of ℓ_1 is greater than the cost of ℓ_3. Both labels will continue selecting the path through the arc with cost 5, and ℓ_3 will end up being the cheapest. The only way for ℓ_1 to recover and become the cheapest would be if the time it arrives at x is so different from the time of ℓ_3 that a drastic change in the weather conditions could be experienced. This seems unlikely, and the results of [8] seem to suggest that it is safe to assume the FIFO property for costs and not to assume it for constraints.

Further, it is possible to define a partial order relation between the constraint sets associated with labels. Intuitively, if ℓ_a has activated the same constraints as ℓ_b but ℓ_b has activated fewer terms than ℓ_a, then we can say that the constraints

[3] Although D contains cycles and although, theoretically, the cycles could be profitable because of the time dependency of costs, we do not allow labels to expand to already visited vertices because routes with cycles would be impractical.

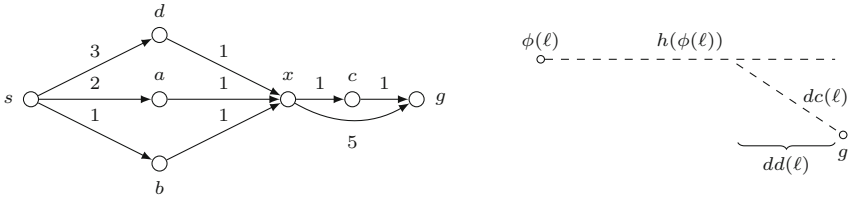

Fig. 1. Left, a domination example. The labels are $\ell_1 = ((s, a, x), 3, \{C_1\})$, $\ell_2 = ((s, d, x), 4, \varnothing)$ and $\ell_3 = ((s, b, x), 2, \{C_1\})$. Right, the setup for the SingleDescent estimate.

of the label ℓ_b are *implied* by those of ℓ_a. We refer to [7] for a precise definition of constraint implication and for implementation details.

Finally, we can state the domination criterion implemented in Lines 19–23. A label ℓ_a is *dominated* by another label ℓ_b if $\phi(\ell_a) = \phi(\ell_b)$, $c_{\ell_a} > c_{\ell_b}$ and Δ_{ℓ_b} is implied by Δ_{ℓ_a}. A label that is dominated is removed from (Line 21) or not inserted into (Line 23) the open list. If $\phi(\ell_a) = \phi(\ell_b)$, $c_{\ell_a} > c_{\ell_b}$ but Δ_{ℓ_b} is *not* implied by Δ_{ℓ_a}, then we say that ℓ_a is only *partially dominated* by ℓ_b. Partially dominated labels cannot be removed from the open list.

4 Heuristics

We present novel heuristics designed to cope with the large sizes of networks representing the 3D problem. We distinguish between two kinds of heuristics: those that implement the estimation of the remaining cost from a node to the goal, a crucial component of A* algorithms, and those that use the desired shape of the vertical profile to add constraints and prune the search space.

Further, we describe a two-phase approach, inspired by the current practice in industry, and we will use that as a benchmark for the heuristic 3D algorithms.

4.1 Remaining Cost Estimation in A*

The estimated cost $\boldsymbol{h}$ is said to be *admissible* if, for any node $u \in V$, the value $h(u)$ does not overestimate the final cost from u to the goal. Further, it is said to be *consistent* if for every node $u \in V$, $h(u)$ is at most the cost of getting to a successor $v \in V$ plus $h(v)$. Consistency can be shown to be the stronger property as it also implies admissibility. If both properties hold, then Algorithm 1 is optimal. If a heuristic is admissible, but not consistent, an optimal solution can be guaranteed if the algorithm allows for more than one label to be expanded from the same node. In standard A* search algorithms, labels expanded from a node are moved to a closed list, which is used to avoid expanding labels again from the same node. Instead, our Algorithm 1 allows more than one label to be expanded from a node in order to handle the constraint-induced lack of FIFO property. The use of an inconsistent rather than a consistent estimate increases the risk of re-expansions occurring and thus lessens the efficiency of the algorithm.

However, guaranteeing consistency, by ensuring that for all arcs $uv \in A$, $h(u) \leq h(v) + c_{uv}$, can be challenging in the FPP setting. First, it is not possible to compute a time-dependent estimate for each node of the network considering all its three coordinates. This would require a full 3D backwards search, which would be computationally heavy, quickly invalidated by the change of data, and pose memory problems. Second, restricting to considering only the 2D positions of the waypoints, say, the projection $\pi(u)$ onto the Earth for any node $u \in V$, one could define an estimate based on the great circle distance from $\pi(u)$ to the goal. But choosing the additional weather conditions would be more challenging: for example, assuming tailwind at cruise level everywhere would not only yield quite optimistic (i.e., loose) estimates but also would not guarantee consistency due to the ignored, possibility of cheaper descent of some levels. Other attempts at defining the flying conditions might lead to violation of admissibility as well.

In this light, with the goal of allowing precomputation of the estimate when a query is received and before the A* algorithm is started, we explore two directions: making the heuristic consistent in spite of worsening its tightness wrt. true cost, and giving up consistency while trying to improve the tightness. In both cases, we strive to maintain estimate admissibility, without which Algorithm 1 would not find the optimal solution. Both estimate procedures contain the following elements. They consider a 2D network obtained by the projection on the ground of all points from V, they associate time and weight independent costs with each arc, and they compute a cheapest path using a backwards Dijkstra from the goal to each node of the 2D network. The estimate $h(v)$ of each node $v \in V$ is then equal to the estimate of the projected node, i.e., $h(v) = h(\pi(v))$. The two procedures differ in the way the resource independent cost is determined.

All Descents: We set the cost of an arc between any two projected nodes to be equal to the cheapest possible cost of going between any pair of nodes that project on those nodes. More formally, let x and y be two nodes in the 2D network and let $\pi^{-1}(x)$ and $\pi^{-1}(y)$ be the set of nodes from $V(D)$ that project to x and y, respectively. We set the cost δ_{xy} between x and y in the 2D network as follows: $\delta_{xy} = \min\{c_{uv} \mid u \in \pi^{-1}(x), v \in \pi^{-1}(y), uv \in A\}$. Clearly, this is the most conservative choice; it guarantees that the derived estimates are consistent for all $u, v \in V$. Indeed, since $h(x) - h(y) = \delta_{xy}$, for any pair of points $u, v \in V$ that project to x and y, respectively, the inequality $h(u) - h(v) \leq c_{uv}$ will be satisfied. We refer to this estimate as AllDescents. The algorithm described in Sect. 3 with the AllDescents estimates is optimal, in the very likely case that the FIFO assumption on costs holds.

Single Descent: While AllDescents is consistent and admissible, its estimates are unrealistically cheaper than the real costs. Indeed, the cheapest arc between any two connected waypoints is very likely a descent from the top-most allowed flight level to the lowest flight level the aircraft can reach. The estimate associated with a node at a given altitude then becomes the cost of the aircraft descending between every two nodes in the shortest path to the destination.

We design an estimate that is closer to the real cost by considering only cruises between nodes. More specifically, between any pair of nodes x and y in the projected network, we consider the cost $\delta_{xy} = \min\{c_{uv} \mid u \in \pi^{-1}(x), v \in \pi^{-1}(y), uv \in A, uv \text{ is a cruise}\}$. It is realistic in most scenarios as the cruising phase is by far the most important and longest phase of a flight and thus the estimated value will often be consistent. However, there are scenarios where it is optimal for the aircraft to descend early on the route, which could lead to inconsistent estimates. Unfortunately, the estimates could also be inadmissible as the aircraft will have to descend at some point to reach the destination airport. To decrease the chance of this occurring, we correct the estimate by including a single direct descent to the goal when calculating the shortest path to the goal. We achieve this by updating the estimate when it is needed. Thus, whenever a label ℓ at $\phi(\ell)$ needs to be evaluated, we subtract from its estimate $h(\phi(\ell))$ the difference between the cost of a descent to the ground and a cruise of the same length using the great circle distance of the potential descent. We retrieve the cost of descending to the ground using the weather and weight conditions of ℓ. We denote this cost $dc(\ell)$ and the distance required for the descent by $dd(\ell)$. From the tables, we then retrieve the cost of cruising for $dd(\ell)$ under the same weather and weight conditions of ℓ. We denote this cost $\delta_{dd(\ell)}$. See Fig. 1, right. Thus, formally, the updated estimate h' for ℓ at $\phi(\ell)$ becomes $h'(\ell) = h(\pi(\phi(\ell)) - (\delta_{dd(\ell)} - dc(\ell))$. We refer to this estimate as SingleDescent.

4.2 Pruning Heuristics

Descent Disregard: Every aircraft has a flight level at which, under average weather conditions, it is optimal for it to cruise. In industry, this flight level is often enforced. We use the information on this *desired flight level* (DFL) heuristically to avoid expanding unpromising labels at low flight levels, thus reducing the search space of the A* search algorithm. We achieve this as follows.

Normally A* tries any possible climb, descent, and cruise along an arc when expanding a label, but many labels created (especially descents and cruises at low and inefficient flight levels) will not lead to optimal solutions. We aim for the algorithm to reduce the number of irrelevant descent and cruise expansions. Using the aircraft performance data we can determine the greatest descent distance (GDD), which is the greatest distance required to be able to reach the ground level from any flight level under any weight and weather conditions. If the distance to the destination airport is greater than the GDD, we do not need to worry about being able to reach the ground in time and can focus on remaining at an efficient flight level. We could therefore disallow descent and cruise expansions for labels not yet at the DFL and outside of the GDD. We still need to worry about constraints that could be blocking the DFL, so we must allow some deviation. We therefore define a threshold for flight levels where, if the aircraft is below this threshold and the distance to the destination is greater than the GDD, descents and cruises are not allowed. The value of the threshold determines a trade-off between missed optimality and speed-up. We have chosen to set the threshold to 3/4 of the DFL, as experiments indicated that this value

yields a large decrease in run-time and only a small decrease in solution quality. We refer to this heuristic as DD.

Climb Disregard: Flight routes generally consist of three phases: climb, cruise, and descent. It is desirable for passengers' comfort to have only one of each of these phases. Our algorithm does not take this into account as it allows for unlimited climbs and descents in order to avoid constraints or to find better weather conditions. Limiting the number of times the aircraft can alternate between climbing and descending will result in more comfortable routes and speed up the algorithm. On the other hand, as the routes will be more constrained, it will lead to an increase in the total cost of the path.

We implement the rule that allows switching only once from climbing to descending, so once a descent has been initiated, no further climbs are allowed. We still allow paths to have a staircase shape, having climbs and descents interleaved with cruising arcs. This could be advantageous when further climbing becomes appealing only after some weight has been lost by consuming fuel.

To monitor the switches we equip the labels with a binary information indicating whether or not a descent has been performed. The label ℓ_a of a path that has already made a descent will be more constrained than the label ℓ_b of a path that has not. Thus, ℓ_a should not be allowed to dominate ℓ_b, even if ℓ_b is more expensive. The decreased effect of domination implies an increase in run-time that may outweigh the reduction of labels expanded due to the restriction. In order to assess the effect of this lack of domination experimentally, we include two configurations in our tests: one where we ignore the switch constraint in domination (thus, ℓ_a would have dominated ℓ_b), and one in which we treat the switch constraint as any other constraint (thus, ℓ_a would be only partially dominating ℓ_b and hence ℓ_b is not discarded). We refer to the former version as CD and to the latter as CDS.

4.3 Two-Phase Approach

In this approach to the FPP, we first find a path through airway points, solving a horizontal 2D routing problem, and then decide the vertical profile of the route by solving a vertical path finding problem.

Solving the horizontal 2D problem, considering only a given altitude at every airway point, does not work well with the second phase, because for many of the routes found, it is then impossible to find a feasible vertical profile. Instead, we "simulate" a horizontal 2D path finding problem by only considering one arc between two airway points when expanding a label, namely the feasible arc whose arrival node is closest to the DFL for the aircraft. Thus, for example, the label at the departure node only expands through arcs going from 0 to 200 nominal altitude; at the next node a label only expands on arcs going from 200 to 400 nominal altitude; and, from there, if 400 is the DFL, the label only expands through arcs to nodes at the same flight level. We only consider descents if it is the only option. Thus, we consider any node, whose projection is the destination airport, as goal nodes.

Climbing and descending decisions are handled in the vertical routing, where we restrict ourselves to the path through the airway points found in the previous phase but reintroduce freedom to determine the altitude. During the search, all possible climbs and descents from each node are allowed.

In both phases, we make use of the same constraint framework as in the 3D solution, iterating the search until a feasible path is found. However, limiting the search in the first phase to only one arc per pairs of waypoints can lead to a route that cannot be made feasible in the second phase, because, for example, of missing links for the descent phase. Hence, when the second phase fails to find a feasible vertical profile, we analyze which combination of arcs caused the problem, introduce a forbidden constraint containing those arcs, and restart the first phase. This can lead to many restarts and even unsolvable instances, as the constraints we introduce may be more severe than they are in reality.

We also include versions of the two-phase approach that use the DD and the CD heuristics. These heuristics address the vertical profile and hence only apply in the vertical optimization phase. We do not vary the estimate heuristics because the conditions here are different from the 3D case: in the first phase, we use an estimate heuristic similar to SingleDescent but without update. In the second phase we do not use any estimate and hence the search corresponds to a Dijkstra search. We were not able to come up with an A* estimate that could be helpful because, with the main focus in this phase being on altitude, estimates tend to be loose and pose computational problems due to the need of calculating these values on the 3D network (or recalculating them every time a new horizontal route is found).

5 Experimental Results

We have conducted experiments on real-life data to compare combinations of the elements presented here. All algorithms used the template of Algorithm 1. In the instantiated versions, the algorithms use the AllDescents or the SingleDescent estimate heuristics combined with reasonable pruning heuristics: DD, CD, CDS, DD + CD, DD + CDS. We also include the versions where AllDescents and SingleDescent are not combined with any pruning heuristic. Finally, we compare with the two-phase approach, which is closest to methods used in practice.

The real-life data is provided by our industrial partner. This data consists of aircraft performance data, weather forecast data in standardized GRIB2 format, and a navigation database containing the information for the graph. The graph consists of approximately 200,000 nodes and 124,000,000 arcs. The aircraft performance data refers to one single aircraft. The data for the weather forecast is given at intervals of three hours on specific grid points that may differ from the airspace waypoints. When necessary, we interpolate both in space and time.

In the calculation of the AllDescents and SingleDescent estimates, we assume piece-wise linearity of the consumptions $\boldsymbol{\tau}$ on the arcs. Consequently, the cheapest cost of an arc can be determined by looking only at the points where measurements are available (e.g., at time intervals of three hours). Under this assumption,

the AllDescents estimate is admissible. Note however that, as shown in [5], the travel time function between data points is not piece-wise linear and so neither is our cost function that includes fuel consumption. To get as close as possible to having an optimal algorithm as *baseline* for the comparison, we therefore included an algorithm without pruning heuristics and with the AllDescents estimate derived from costs on arcs given by a piece-wise linear function, whose pieces in the time scale are reduced to five minutes intervals. This approach would not be feasible in practice, as it took more than 40 h in our computational environment (see below) to precompute the estimates.

A test instance is specified by a departure airport and time, and a destination airport. A set of 13 major airports in Europe was selected uniformly at random to explore a uniform coverage of the constraints in the network. Among the 156 possible pairings, 16 were discarded because of short distances, resulting in 140 pairs that were used as queries. The great circle distances of these instances range from 317 to 1682 nautical miles. All algorithms were implemented in C# and the tests were conducted on a Dell XPS 15 laptop with an Intel Core i7 6700HQ at 2.6 GHz with 16 GB of RAM. Each algorithm was run 3 times on each instance and only the fastest run was recorded. All runs had a time limit of one hour.

We visualize the results in Fig. 2. The plots are disposed such that column-wise we distinguish the estimate heuristic and row-wise the performance measure considered. The pruning heuristics are represented along the x-axis of each plot. The quality of a route is its monetary cost. We show the percentage gap with respect to the results found by the baseline algorithm (5 min pieces). The run-time is the time spent for preprocessing (i.e., calculating the estimates) plus the total time spent searching. The preprocessing times for any configuration (apart from the baseline algorithm, which is not shown) are almost identical, so we do not distinguish. Time is measured in seconds and a logarithmic transformation is applied. We did not observe a clear dependency of the time on the mile distance of the instances, hence this latter is not visualized. Points represent the results of the selected runs. Points of results attained on the same instance are linked by a gray line. The boxes show the first, second (median) and third quartile of the distribution.

Comparison: The first observation is that the gap of AllDescents without pruning heuristic is different from zero only in a couple of instances and only in one making the gap worse than 1.5%. Hence, in practice assuming piece-wise linearity of consumptions seldom deteriorates the results. However, this observation might be dependent on our setting, as Blanco et al. [5] do show a relevant impact. Comparing AllDescents with SingleDescent, we observe that the inadmissibility of SingleDescent does not worsen the solution quality in any instance while wrt. run-time, SingleDescent leads to a considerable save for most instances. Tighter estimates allow for more restrictive search; a narrower area around the optimal route. We give evidence of this in Fig. 3, where we show the search results for one instance using the two estimate heuristics. The figure depicts the horizontal section of the 3D network restricted to the arcs that were actually expanded

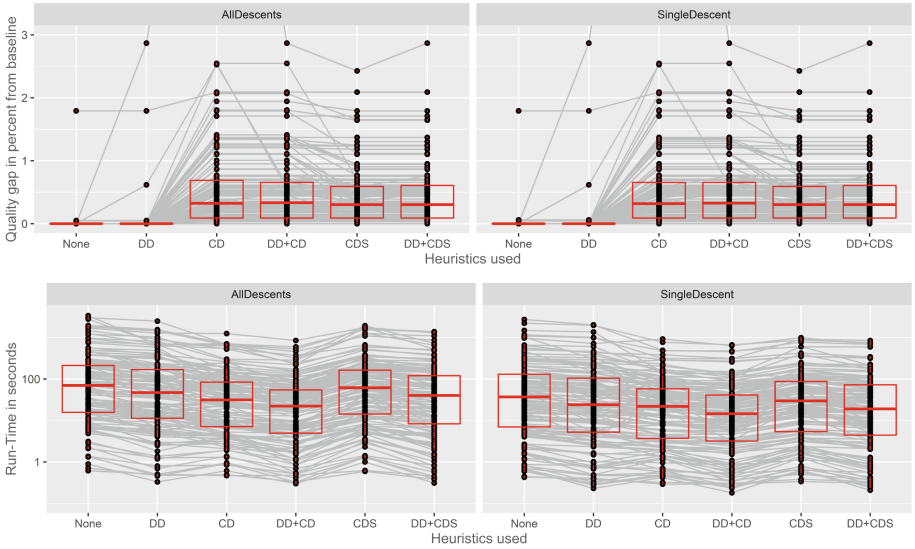

Fig. 2. Solutions quality (top) and run-time (bottom) results on 140 queries.

during the search. SingleDescent is much better at narrowing the search compared with AllDescents. However, the figure does not show the vertical dimension and the several flight levels expanded from nodes nor the arcs that were expanded more than once in SingleDescent. There are 14 instances where the extra expansions to cope with the inconsistency outweigh the expansions saved from the tighter estimate and make SingleDescent slower than AllDescents.

Considering pruning heuristics, DD leads to worse solution quality in only 5 instances for both estimate heuristics while improving the run-time for both. As expected, solutions become more costly when including heuristics that restrict the vertical profiles, CD and CDS. However, because these routes are more stable, they are more comfortable, a quality component not included in the monetary cost otherwise considered. If we include correct domination, as done in CDS, the solution quality improves slightly over algorithm versions that use CD, but considering the run-time, CDS is computationally more demanding than CD. In general, profile-based heuristics have considerable impact on the run-time. The CD and DD + CD heuristics contribute most to the speed-up. In a few instances, however, they can perform worse than the variants with no pruning heuristic. In these cases, finding a different solution (which might violate a constraint) can cause the algorithm using CD to do extra passes. Finally, the configuration using both DD and CD is the fastest. It has a median run-time slightly above 14 s and a worst-case performance of 14 min.

Comparison with the Two-Phase Approach: The two-phase approach with no pruning heuristics has a median quality gap of 1.9%, which is significantly higher than any of the 3D algorithms. There are also a few heavy outliers with

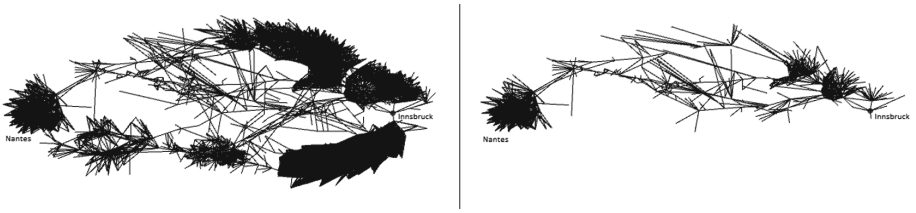

Fig. 3. Comparison of arcs expanded by AllDescents (left) and SingleDescent (right) with no heuristics on a route from Innsbruck to Nantes.

5 instances having a gap over 10%. In these instances, the two-phase approach returns a path through arcs where flying at the desired flight level is forbidden by some constraints and thus the vertical optimization will not be able to repair and obtain a good quality solution. There are also two instances where the two-phase approach fails to find any solution at all. As far as run-time is concerned, the two-phase approach does entail faster run-times than any of the 3D algorithms with a median of 10 s and a worst-case performance of 47 min. In general, there are some outliers caused by instances where the vertical optimization has trouble finding a feasible solution causing a large number of restarts. Using the DD heuristic does not yield any relevant effect, while using the CD heuristic does decrease run-time but also worsens the cost of the final path further. CD also increases the number of unsolved instances to 10.

6 Concluding Remarks

We found the SingleDescent estimate together with the DD heuristic particularly interesting. With respect to the near optimal solution of the baseline algorithm, this algorithm leads to deterioration of solution quality in only a few instances while it provides a considerable decrease in run-time. If further speed-up is needed, the CD heuristic could be added at the cost of only a slight increase in deterioration. The routes attained adopting the CD heuristic might be preferable in practice anyway due to the more stable vertical profile.

Perhaps our most important contribution is that we demonstrate the practicability of a direct 3D flight planning approach. Our comparison against the two-phase approach, more commonly used in practice, shows that although median results for the latter are better in terms of run-time, the solution quality and the outliers in run-time and unsolved instances are considerably worse than the results attainable with direct 3D approaches. The 3D approach exhibits fewer outliers, terminates on every instance, and is therefore more robust. If run-time is really an issue, we have shown that by combining the SingleDescent, the DD, and the CD heuristics, competitive time performance can be achieved, while retaining superior robustness and smaller reduction in solution quality relative to the two-phase approach. The ideas tested here will be included in future releases of software from our industrial partner.

References

1. Batz, G.V., Geisberger, R., Sanders, P., Vetter, C.: Minimum time-dependent travel times with contraction hierarchies. ACM J. Exp. Algorithmics **18**(1), 1.4:1–1.4:43 (2013). Article no. 1.4
2. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik **1**(1), 269–271 (1959)
3. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. Syst. Sci. Cybern. **4**(2), 100–107 (1968)
4. Yinnone, H.: On paths avoiding forbidden pairs of vertices in a graph. Discrete Appl. Math. **74**(1), 85–92 (1997)
5. Blanco, M., Borndörfer, R., Hoang, N.-D., Kaier, A., Schienle, A., Schlechte, T., Schlobach, S.: 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). OpenAccess Series in Informatics (OASIcs), vol. 54, pp. 12:1–12:15. Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2016)
6. Bast, H., Delling, D., Goldberg, A., Müller-Hannemann, M., Pajor, T., Sanders, P., Wagner, D., Werneck, R.F.: Route planning in transportation networks (2015). arXiv:1504.05140 [cs.DS]
7. Knudsen, A.N., Chiarandini, M., Larsen, K.S.: Constraint handling in flight planning. In: Beck, J.C. (ed.) CP 2017. LNCS, vol. 10416, pp. 354–369. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66158-2_23
8. Knudsen, A.N., Chiarandini, M., Larsen, K.S.: Vertical optimization of resource dependent flight paths. In: 22nd European Conference on Artificial Intelligence (ECAI). Frontiers in Artificial Intelligence and Applications, vol. 285, pp. 639–645. IOS Press (2016)
9. Blanco, M., Borndörfer, R., Dung Hoàng, N., Kaier, A., Casas, P.M., Schlechte, T., Schlobach, S.: Cost projection methods for the shortest path problem with crossing costs. In: D'Angelo, G., Dollevoet, T., (eds.) 17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS). OpenAccess Series in Informatics (OASIcs), vol. 59, pp. 15:1–15:14. Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2017)
10. Jensen, C.K., Chiarandini, M., Larsen, K.S.: Flight planning in free route airspaces. In: D'Angelo, G., Dollevoet, T., (eds.) 17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS). OpenAccess Series in Informatics (OASIcs), vol. 59, pp. 14:1–14:14. Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2017)