Vertical Optimization of Resource Dependent Flight Paths

Anders N. Knudsen and **Marco Chiarandini** and **Kim S. Larsen**[1]

Abstract. Flight routes are paths calculated on a network of waypoints representing 3D-coordinates. A common approach is first to calculate a path in a 2D-network, taking into account feasibility constraints, and then to optimize the altitude of the flight.

We focus on the problem of determining the vertical trajectory defined by an altitude at each waypoint of a 2D-route. The cost of an airway depends, directly, on fuel consumption and on flying time, and, indirectly, on weight and on weather conditions. These dependencies invalidate the FIFO property, which is commonly assumed for time-dependent networks. Moreover, the amount of fuel at departure has an impact on the weight and depends on the length of the route. This, therefore, needs to be decided upon for our problem. We aim at minimizing the total cost.

We study path-finding algorithms, both exact and heuristic, that we iterate in a line-search procedure to decide the fuel amount at departure. We use real-life data for the experimental analysis and conclude that on those data the assumption of the FIFO property remains a good heuristic choice.

1 Introduction

Flight route optimization aims at finding 3D-paths for aircraft in airway networks that minimize the total cost determined by fuel consumption and flying time. It has evident financial and environmental impacts. Airway networks can be huge, due to the added dimension compared with road networks, and side constraints complicate further the problem. Moreover, most of the constraints are determined by a central control institution, e.g., Eurocontrol in Europe and FAA in USA, and change rapidly with time in order to take traffic conditions into account. Therefore, the common practice is to determine the precise flight route a few hours before take-off. For this to be feasible and bring any advantage, the route determination must be very fast, say on the order of a few seconds. If necessary, the route can then be adjusted during the flight by real-time optimization, considering more up-to-date information.

In this work, we focus on the off-line flight route optimization. The common approach in the industry has been to decompose the problem: first a shortest, feasible 2D-route is found and, afterwards, the vertical trajectory is optimized. The problem consists in finding a shortest path in a network, made of nodes that are the combination of waypoints, decided by the 2D-route returned by the first stage, and the allowed altitudes for the aircraft. Arcs connect nodes belonging to consecutive waypoints in the 2D-route if the distance between them is enough for the aircraft to make the corresponding altitude transition.

The costs of flying through these arcs depend on two resources: time and fuel. There is a direct dependency because airlines calculate the total cost as a weighted sum of time and fuel consumed and there is an indirect dependency, because the amount of time and fuel consumed depends on the performance of the aircraft, which is influenced by the weather conditions, which in turn depend on time, and by the weight, which in turn depends on the fuel consumed. A consequence of these dependencies is that the cost and the feasibility of each arc are not statically given but become known only when the path to a node is determined. The situation is further complicated by the fact that the amount of fuel at departure is not given, since it is also a decision that can be optimized. The main issue for solution algorithms is that due to the indirect dependencies and the impossibility to wait at the nodes (even at the departure airport), the total cost is a non-monotone function with respect to time and fuel. That is, it may be disadvantageous to arrive cheaply at a node (and hence earlier or lighter than other alternatives) as it may preclude the chance to obtain high savings in the remaining path due to favorable weather conditions developing somewhere at a later time. For labeling algorithms, this means that a total ordering of the labels at the nodes is not available.

In the latest years, a considerable amount of research has focused on engineering aspects of algorithms for the shortest path problem (SPP) on large road networks [1]. Huge improvements have been achieved using several advanced techniques, most of which we deem inapplicable in our setting due to arc costs being unknown before starting the search. In time-dependent SPP arc costs can change with the time of traversal. In one of the first solution attempts to solve this variant, Bellman's iterative algorithm [3] was extended by performing finitely many iterations for any possible starting time [5]. A property of the problem that demarcates solution approaches is the FIFO property. If a network is FIFO, then a vehicle leaving a node cannot be overtaken by another vehicle leaving the same node at a later stage. The SPP in time-dependent FIFO networks is polynomially solvable [9], while

[1] Department of Mathematics and Computer Science, University of Southern Denmark, Denmark
email: {andersnk,marco,kslarsen}@imada.sdu.dk
The first author acknowledges financial support by the Innovation Fund Denmark. The third author was partially supported by the Danish Council for Independent Research, Natural Sciences, grant DFF-1323-00247.

the problem becomes NP-hard for non-FIFO networks [12]. A variation of the FIFO property, which we will continue to refer to as FIFO, can be formulated also for the vertical trajectory problem that we study here. However, in general this property does not hold for this problem. Most of the literature has focused on FIFO networks. In [4], the authors present an extension for FIFO networks that results in significant speed-up over earlier Dijkstra approaches. In [6], a lower bound on the cost from each node to the goal, for use as the heuristic in A*, is calculated in a preprocessing step by backwards breadth-first search assuming the fastest possible speed on each arc. In [2], travel time profile queries are discussed, where the departure time at the source can be shifted at convenience, in a similar fashion as we can decide the amount of fuel to embark at departure. The shortest path with time-dependencies in non-FIFO networks has so far been disregarded in the literature. Articles discussing a problem similar to ours appeared only in specialized literature (see e.g., [13]).

We study the classical shortest path methods, breadth-first[11][10], Dijkstra[7] and A*[8] search, modified to take resource dependencies into account. We compare their performance in terms of quality and running time. In particular, we set out to assess empirically whether it is important in terms of final cost to work without the FIFO assumption and whether it is computationally feasible. We design a lower bound for A*. Under the FIFO assumption we show that this A* search runs faster than Dijkstra's algorithm while still producing optimal solutions under the assumption. Then, we show that the A* approach in [6] but without the FIFO assumption leaves the search computationally infeasible. To improve running time we introduce an upper bound to prune path extensions. We are able to conclude that for the real life data, on which we based our study, the increased scrutiny obtained by relaxing the FIFO assumption does not pay off.

Our work is in collaboration with the industrial partner Aviation Cloud (AC) A/S, a Danish company, whose core business is in flight route planning. The company is interested in an algorithm that can solve the problem very fast, within a few seconds, since it must be used as an aiding tool for flight route planning in an interactive setting. Differently from other sources, e.g., [13], that used the Base of Aircraft Data, an open source database provided by Eurocontrol, our experimental analysis is conducted on data from the specific business case of Aviation Cloud. As common in the sector, aircraft manufacturers supply airlines with look-up tables to calculate the resource consumption of operating the aircraft. Airlines then provide these data to Aviation Cloud. We include a comparison of the quality of our vertical trajectories with the solution currently in use at Aviation Cloud.

2 Problem formulation

Airspaces in different areas of the world are represented by 2D networks, i.e., directed graphs $D_{2D} = (V_{2D}, A_{2D})$. The nodes in V_{2D} represent *waypoints* defined by latitude and longitude coordinates. Altitude is not part of a waypoint description. The arcs in A_{2D} represent feasible connections between waypoints. A *2D (flying) route* is an (s, g)-path in D_{2D} represented by n waypoints plus a departure node (source) s and an arrival node (goal) g, that is, $R = (s, r_1, \ldots, r_n, g)$, $s, r_i, g \in V_{2D}$ for $i = 1..n$, $(r_i, r_{i+1}) \in A_{2D}$ for $i = 1..n-1$ and $(s, r_1), (r_n, g) \in A_{2D}$.

In this work we assume that a 2D route R is given and we want to optimize the vertical trajectory. Aircraft may only cruise at specific (standard) *flight levels* that may differ from area to area. Hence, at each waypoint r_i, $i = 1..n$, of R we associate a set of flight levels $F_1, \ldots, F_n$. Thus, for a 2D route R a vertical trajectory is an (s, g)-path in a *layered*, directed graph $D_R = (V_R, A_R)$ where $V_R = \{s\} \cup F_1 \cup \ldots \cup F_n \cup \{g\}$ and $A_R = A_0 \cup A_1 \cup \ldots \cup A_{n-1} \cup A_n$ with $A_0 = \{(s, f) \mid f \in F_1\}$, $A_n = \{(f, g) \mid f \in F_n\}$, $A_i = \{(f, h) \mid f \in F_i, h \in F_{i+1}\}$ for $i \in \{1..n-1\}$. If we denote by m the size of all F_i for $i = 1..n$, then $|V_R| = nm + 2 = \Theta(nm)$ and $|A_R| = (n-1)m^2 + 2m$.

Connections in A_R exist if allowed by regulations and by the operational properties of the aircraft. Aircrafts can only begin a climb or a descent at waypoints but they cannot climb or descend more than at a given rate, which depends on the current weight and weather conditions.

For these reasons the existence of connections in D_R can be determined only during the search for routes when the state of the aircraft at a node becomes known.

We consider fuel and time as resources whose amounts are nonnegative, i.e., they belong to $\mathbb{R}_+$. The resource consumption accumulated at a node u in V_R along an (s, u)-path P in D_R is a pair $\vec{\tau}_P = (\tau_P^x, \tau_P^t) \in \mathbb{R}_+^2$, where the superscripts x and t are used to denote the fuel and time components of the resource cost. We assume known the departure time from the starting airport, τ_s^t, and unknown the amount of fuel at departure, τ_s^x.

The changes in the resource consumption associated with an arc $(u, v) \in A_R$ are given by *resource extension function* (REF) values and can be represented by the vector $\vec{f}_{uv} = (f_{uv}^x, f_{uv}^t) \in \mathbb{R}_+^2$. The components of a REF vector, f_{uv}^x and f_{uv}^t, depend on the consumption of resources at the tail node u of arc (u, v), which can be calculated from the resource consumption accumulated along a path P from s to u. Specifically, the fuel at u is $\tau_s^x - \tau_P^x$ and the time is $\tau_s^t + \tau_P^t$. The resource consumption accumulated along a path $P' = (s, \ldots, u, v)$ that extends P by an arc (u, v) is thus $\vec{\tau}_P + \vec{f}_{uv}(\vec{\tau}_P)$.

For a given aircraft, REF values are looked up in a set of tables of *aircraft performance data* that output the time and fuel consumption for an arc using as inputs: (i) flight level of starting and arrival point, (ii) weight, (iii) temperature deviation, (iv) wind component, and (v) cost index. The cost index, ρ, is an efficiency ratio between the time-related cost and the fuel cost that airlines use to specify how to operate the aircraft. It determines the speed of the aircraft and we assume it given. It is decided at strategic level and it cannot be changed during the planning phase. Note that for a path $P = (s, \ldots, u)$ and an arc $(u, v) \in A_R$, $\vec{f}_{uv}$ depends on τ_P^x because of (ii) and depends on τ_P^t because of (iii) and (iv).

A path $P = (s, v_1, \ldots, v_k)$ in D_R is *resource feasible* if $\tau_{P'}^x > 0$ for any prefix path of P' (there are no restrictions for $\tau_{P'}^t$). Both resources contribute to define the cost function $c : A_R \times \mathbb{R}_+^2 \to \mathbb{R}_+$ for the network specified by D_R. This function represents the monetary cost of flying through an arc. Airlines use the following formula (see e.g. [13]):

$$c((u, v), \vec{\tau}_P) = f_{uv}^t(f_{uv}^x / f_{uv}^t + \rho).$$

For an arbitrary path $P = (v_1, \ldots, v_k)$ in D_R with prefixes $P_i = (v_1, \ldots, v_i)$, $i = 2..k-1$, its resource-dependent cost, c_P,

is defined recursively as follows:

$$\begin{aligned} c_{P_1} &= 0 \\ c_{P_i} &= c_{P_{i-1}} + c((v_{i-1}, v_i), \vec{\tau}_{P_{i-1}}) \end{aligned}$$

We can now, more formally, define our vertical flight trajectory optimization problem.

Definition 1 (Vertical flight trajectory problem)**.** *Given a 2D flight route R from a departure node s to an arrival node g, the layered digraph $D_R = (V_R, A_R)$ constructed from R and the available flight levels, a resource extension function $\vec{f} : A_R \to \mathbb{R}^2$, a cost function $c : A_R \times \mathbb{R}^2_+ \to \mathbb{R}_+$, and a departure time τ^t_s at s; find an amount of fuel at departure, τ^x_s, and a resource feasible (s, g)-path P in D_R such that c_P is minimum, i.e.,*

$$c_P = \min\{c_Q \mid Q \text{ is an } (s,g)\text{-path in } D_R \text{ and } \tau^x_{Q'} > 0 \text{ for any prefix } Q' \text{ of } Q\}.$$

Further we define two functions $\lambda, \mu : A_R \to \mathbb{R}_+$ with the following properties:

$$\lambda(u,v) \le c((u,v), \vec{\tau}_P) \quad \forall\, (u,v) \in A_R,\ \vec{\tau}_P \in \mathbb{R}^2_+, P = (s, \ldots, u) \subseteq D_R$$

$$\mu(u,v) \ge c((u,v), \vec{\tau}_P) \quad \forall\, (u,v) \in A_R,\ \vec{\tau}_P \in \mathbb{R}^2_+, P = (s, \ldots, u) \subseteq D_R$$

In other terms, λ is a lower bound and μ an upper bound on the cost of arcs independent on resources. In our application, the minimal cost can be given by any condition from having the strongest tailwind and carrying as little fuel as possible to having the strongest headwind and carrying the maximum amount of fuel.[2] The values of λ and μ for every arc $uv \in A_R$ are calculated in a global preprocessing phase and saved in a table. More precisely, for each arc, we look up the corresponding distance and flight level change in the aircraft performance data and try all conditions recording the smallest and highest cost. Distances are discretized and the value to look up is rounded down for calculating λ and up for μ.

Using these functions, it is possible to calculate a lower and an upper bound on the cost of a path from any node $u \in V_R$ to the final node g, i.e.,

$$LB(u,g) = \min\left\{ \sum_{i=1}^{n} \lambda(v_i, v_{i+1}) \mid (v_1, \ldots, v_{n+1}) \text{ path in } D_R \text{ with } v_1 = u, v_{n+1} = g \right\}$$

$$UB(u,g) = \max\left\{ \sum_{i=1}^{n} \mu(v_i, v_{i+1}) \mid (v_1, \ldots, v_{n+1}) \text{ path in } D_R \text{ with } v_1 = u, v_{n+1} = g \right\}$$

[2] Indeed, although counterintuitive, high weight may imply low cost, because it may allow descents at a speed otherwise impossible. Similarly, strong headwind may allow an airplane to climb or descend larger differences in altitude for a given ground distance than under normal conditions.

These values can be computed in a preprocessing stage by a backwards breadth-first search.

An analogous of the FIFO property (or non-overtaking property) for time-dependent networks can be defined for our case as follows.

Definition 2 (FIFO property)**.** *For any pair of nodes, $u, v \in V_R$, and any pair of paths, $P = (s, \ldots, u)$ and $Q = (s, \ldots, u)$, arriving in u with $\vec{\tau}_P$ and $\vec{\tau}_Q$, respectively, if $c_P < c_Q$ then $c_P + c((u,v), \vec{\tau}_P) \le c_Q + c((u,v), \vec{\tau}_Q)$.*

However, this property may be violated in our case. Indeed, if at node u an (s,u) path Q is more expensive than an (s,u)-path P ($c_P < c_Q$) then, because of the way the cost is calculated, it must be that Q has consumed more time or more fuel or both. But in all these cases it is still possible that there is a cheaper $(s, \ldots, u, v)$ path extending Q rather than P if, for example, weather conditions at time $\tau^t_s + \tau^t_Q$ are more beneficial than at time $\tau^t_s + \tau^t_P$ (and waiting is not possible). Although the FIFO property seems not to hold in theory it is unclear how much it is violated in the real data and how much worse we do by assuming it as a heuristic. We study therefore both the FIFO and the non-FIFO cases.

3 Algorithm design

The overall solution approach is sketched in Alg. 1. The main function, FINDFUELANDPATH(), performs a line search on the amount of fuel at departure. We start with an initial guess $\tau^x_{s,0}$. Then at each iteration i, we call a path finding procedure, FINDPATH, with the current guess $\tau^x_{s,i}$. If the amount of fuel is sufficient to find a path in D_R from s to g such that $\tau^x_{g,i} \ge 0$, then we update our guess with $\tau^x_{s,i+1} = \tau^x_{s,i} - \tau^x_{g,i}$. Otherwise, the path finding procedure continues with empty tank until the goal is reached accumulating a negative $\tau^x_{g,i}$, which is used to update $\tau^x_{s,i+1}$ yielding an increase of value. We continue in this way until the tentative values converge, that is, when the amount of fuel carried, but not spent, $\tau^x_{g,i}$, is less than 2% of the fuel at departure, $\tau^x_{s,i}$. Depending on the discretization of the aircraft performance data, the algorithm might never reach this state. This can be counteracted by increasing the threshold after several iterations. However, this was not necessary in any of our tests.

The core of the FINDFUELANDPATH function is the path finding function FINDPATH, which will be executed a few times. We focus, therefore, on algorithms to make this function effective and efficient. We consider breadth-first search and variants of best-first search. In Alg. 1 we give a general template that remains valid for all the algorithms described below. A *label* is a piece of information associated with a node of D_R and created when the algorithm first considers or "opens" a node. The expansion of a label is the operation of generating a new label for any node in D_R reachable by an arc going out from the node of the label under expansion.

The algorithm takes as input information about how to construct the network, the REF tables, the amount of fuel at departure and the departure time. It then works with a list of open labels $\mathcal{Q}$ and in turn expands labels from this list. The specific algorithms differ by how a label is selected for expansion (line 13) and by how it is inserted in the list (line 20).

1 **Function** FINDFUELANDPATH($D_R, \vec{f}, c, \tau_s^t$)
2 initialize $\tau_{s,0}^x$
3 $P, \tau_{g,0}^x, \tau_{g,0}^t \leftarrow$ FINDPATH($D_R, \vec{f}, c, \tau_{s,0}^x, \tau_{s,0}^t$)
4 $i \leftarrow 0$
5 **while** $\tau_{g,i}^x \geq 0.02 \cdot \tau_{s,i}^x$ **do**
6 $P, \tau_{g,i}^x, \tau_{g,i}^t \leftarrow$ FINDPATH($D_R, \vec{f}, c, \tau_{s,i}^x, \tau_{s,i}^t$)
7 $\tau_{s,i+1}^x = \tau_{s,i}^x - \tau_{g,i}^x$
8 $i = i + 1$
9 **return** P, τ_s^x

10 **Function** FINDPATH($D_R, \vec{f}, c, \tau_s^x, \tau_s^t$)
11 initialize the open list $\mathcal{Q}$ by inserting a label for the departure node
12 initialize ℓ with an empty path of cost infinite
13 **while** *true* **do**
14 $\ell' \leftarrow$ select a label from $\mathcal{Q}$ ▷ Selection criterion depends on algorithm
15 **if** *(cost of ℓ' greater than cost of ℓ)* **then break**
16 **if** *(ℓ' is at destination) and (cost of ℓ' smaller than cost of ℓ)* **then**
17 $\ell \leftarrow \ell'$
18 **break**
19 **foreach** *reachable and allowed node at next waypoint* **do**
20 calculate cost of reaching node
21 add new label to $\mathcal{Q}$ ▷ Insertion depends on algorithm
22 **return** P, τ_g^x, τ_g^t *of* ℓ

Algorithm 1: A general template for solving the flight trajectory problem

When expanding a label on line 18 the possible reachable nodes in the next layer are calculated. The distance between consecutive waypoints is given by the 2D-route. It is then possible to calculate which flight levels in the next layer can be reached from a node by iteratively trying higher or lower altitudes until the distance available is not enough for the climb or descent. Using the aircraft performance data one can retrieve the REF vector for each of the reachable altitudes and for each node create a label. Since aircraft are allowed to reach waypoints outside of standard flight levels, we might have to add labels at nodes that are not associated with a standard flight level. These labels are *flagged*, meaning that they can be expanded only by a climbing or descending arc and not by a cruising arc. Only two of these labels need to be created: one at the maximum climb rate and one at the maximum descent rate. Due to the discretization of the performance data, we allow only one flagged label of each kind between each pair of adjacent standard flight levels. The influence of these on the graph is discussed in the analysis section.

Initial fuel amount The initial fuel, $\tau_{s,0}^x$, which is used as the first value in the line search, is calculated as follows. We start with the maximum load and adjust its value using the information gathered by a linear time greedy algorithm. This algorithm expands only one label at each waypoint. The label expanded belongs to the next waypoint and has flight level equal to the minimum between a given value h and the highest flight level reachable given the current condition of the aircraft. Whenever from a waypoint the destination node cannot be reached at the steepest descent, the construction backtracks to the previous waypoint and descends directly to the destination from there. After such construction the amount of fuel at departure is updated. The whole procedure is repeated decreasing h through all standard flight levels and halting when either 5 iterations are done or the fuel amount left after the path is less than 2% of the value at departure. Finally, $\tau_{s,0}^x$ is set equal to the value that generated the cheapest route during this phase.

Breadth-first search All labels at one layer of D_R (corresponding to all the different flight levels at one waypoint of the 2D route) are expanded before moving on to the next waypoint. The order of expansion is not relevant. Using the FIFO property, label domination occurs at each node: for multiple labels reaching a node only the cheapest one is kept. We denote this baseline algorithm by `F-BFS`.

Dijkstra The open list is sorted according to the current cost of the labels and the cheapest label is selected for expansion. The FIFO property is used for label domination and an additional list of closed nodes prevents expanding nodes more than once. We denote this algorithm by `F-Dijk`.

A* The cost of labels is given by the sum of the cost of the path to the corresponding node plus a heuristic cost of the path from the node to the goal. The cheapest label is selected for expansion and, under the FIFO assumption, at any given node, the cheapest label dominates the others, which are then removed from (or never added to) the open list. To guarantee optimality the heuristic must be admissible, i.e., it never overestimates the cost of reaching the goal, and consistent, i.e., for every node u, the estimated cost of reaching the goal from there is no greater than the cost of getting to a successor v plus the estimated cost from v to the goal. It is easy to show that consistency is a stronger property as it implies admissibility. We use the lower bounds LB defined in Sec. 2 that is thus both consistent and admissible. We denote this algorithm `F-A`*.

Non-FIFO A* Lifting the FIFO assumption in `F-A`* means that labels cannot be dominated at nodes based purely on the cost of reaching that node. This algorithm, denoted by `nF-A`*, does not use the FIFO assumption but it tries to reintroduce some form of domination at nodes. It does so using the upper bound UB (see Sec. 2). Let P and Q be two paths in D_R from s to u and let ℓ and ℓ' be the corresponding labels in u, respectively. The label ℓ' is dominated by ℓ, if $c_Q > c_P$ and $c_Q - c_P > UB(u, e) - LB(u, e)$. That is, it is impossible for the most expensive label ℓ' to gain the difference in cost over the cheapest label ℓ in the remaining path to destination even if in the case that ℓ' continues in the best possible way and ℓ in the worst possible way. When a new label is created it may either be dominated by the currently cheapest label on that node, or it may dominate any number of the more expensive current labels at the node. Selection is still based on the cost of the path plus the heuristic cost to the goal.

A* with an inconsistent and inadmissible heuristic Experimental analysis shows that `nF-A`* lacks efficiency. The heuristic via lower bound LB is quite weak and this is bad when domination can occur only seldom. Further experimental observations suggested that using always maximum tail

wind and zero fuel in the calculations of the λ values associated to each arc led to a stronger LB bound and therefore to faster searches. However, as explained earlier in this way we cannot guarantee the consistency and admissibility of the heuristic. However in order to be able to solve instances of a considerable sizes, we introduce a variant of `nF-A*`, which uses the stronger bound. We denote it `nF-iA*`. In Sec. 5 we analyze the trade-off between computation speed and quality of the solutions in the two variants.

We also tested the impact of the stronger lower bound in a FIFO setting. `F-iA*` is a variant of `F-A*` that uses the stronger bound, but also allows a node to be revisited, should a cheaper label be discovered later. The point of allowing revisits is that it causes inconsistency to no longer lead to suboptimal results. This limits the amount of instances where the algorithm does not find optimal solutions. However, it does not fix problems caused by inadmissibility, so optimality is still not guaranteed.

4 Analysis

Data structures The open list is implemented as a red-black tree. We maintain a pointer to the minimum cost label, so we are able to retrieve it for expansion in amortized constant time. Access is worst case constant time and rebalancing after deletion is amortized constant time. To check domination, when a new label is created, the label must be compared with the minimum cost label at the same node. For `F-Dijk` and `F-A*`, there can only be a single label for each node and so they can be stored in a simple list. For `nF-A*`, we need to keep track of the labels per node and be able to retrieve the minimum and the maximum cost labels. This is achieved by keeping a red-black tree of labels at each node with a pointer to the minimum cost label and one to the maximum. The entries in these trees have pointers to the corresponding entries in the open list and vice versa. Then we are able to perform all extractions and deletions in the two structures in amortized constant time. Finally, the closed list is implemented as a bitwise array with one bit per node.

FIFO algorithms Let n be the number of waypoints and m the number of standard flight levels. Due to the domination in `F-BFS`, `F-Dijk`, and `F-A*`, we can expand exactly one label for each node at a standard flight level. Hence, the size of the open list and the number of labels we can expand is $O(nm)$. Each expanded label can generate a maximum of m additional labels. This means that we can create $O(nm^2)$ labels overall. This is also true when taking into account the climb- and descent-only labels. Whenever expanding a label leads to the creation of either, it means that some number of standard labels were not created. The expansion of the climb- or descent-only label can then generate exactly that number of additional labels, leading to a maximum of $m+2$ labels created per expansion if both kinds are created. Due to allowing revisits the labels created by `F-iA*` may increase exponentially to $O(m^n)$.

Upon creation, a label must be inserted into the open list and into the node list. `F-BFS` simply inserts it into a simple list in $O(1)$. `F-Dijk`, `F-A*` and `F-iA*` insert them into the red-black tree that implements the open list in $O(\log(nm))$. With our datasets this cost is, however, dominated by the constant time needed to expand a label, due to several look-ups to calculate the cost of an arc.

Non-FIFO Since domination is restricted in both `nF-A*` and `nF-iA*`, the number of labels in the open list grows exponentially with respect to the depth of the search, which we know to be n. Thus, the open list can end up containing $O(m^n)$ labels. The insertion of each label costs $O(n\log m)$.

Preprocessing The A* algorithms need the values $LB(u, g)$ for all $u \in V_R$. Since these values are resource independent we calculate them once for every (s, g)-path query, that is, before running findPath and hence it is unaffected by the repetition of findPath. We do this by a backwards breadth-first search in $O((n \cdot m) \cdot (n + m))$. For `nF-A*` we also calculate $UB(u, e)$ for all $u \in V$. This latter needs additional $O((n \cdot m) \cdot (n+m))$ for a backwards breadth-first search.

5 Experimental Assessment

All algorithms described above except `nF-A*` are heuristic algorithms, that is, because of the FIFO assumption or the use of inconsistent and inadmissible heuristics they do not guarantee to terminate with optimal solutions. We set out then to assess the deterioration in quality of the heuristic algorithms and their running time with the goal of suggesting the best trade off. We also compare the quality of our solutions with those found by `AC`, a fast greedy algorithm previously in use at Aviation Cloud.

We use real life data provided by Aviation Cloud. This data consists of aircraft performance data, weather data forecast in standardized GRIB2 format and 2D-routes provided by Aviation Cloud's route planner. The aircraft performance data refers to one single aircraft and the standard flight levels are from 0 to 48000 feet in intervals of 200 feet. The weather data forecast are from three different days with intervals of three hours and with varying top wind speeds (the strongest wind speed recorded was 270 km/h). A specific test instance (or query) is determined by the time of departure and the 2D-route. The same aircraft performance data and weather conditions are shared by several instances. All tests were run with a default of 5% contingency fuel. To account for fluctuations in CPU time measurement, each instance was tested 15 times and only the fastest was recorded.

The instances have been grouped by the number of waypoints in the 2D-routes, varying from 11 to 50 with increments of 1. This is the most important factor for the complexity of the problem. For each different number of waypoints 30 routes are available, arbitrarily chosen among the routes serviced by Aviation Cloud. Overall we tested on 3600 instances, which should be enough to guarantee that the visual differences between algorithms in the following analysis are statistically significant.

All algorithms were implemented in C# and the tests were carried out on a virtual machine in a cloud environment with an Intel Xeon E5-2673 processor at 2.40Ghz and with 7GB RAM. We introduced time limits for the non-FIFO algorithms that were 30 seconds for `nF-iA*` and 10 minutes for `nF-A*`. `nF-A*` was able to solve instances only with less than 14 waypoints. FIFO A* algorithms used in the worst case 100 MB of memory of which 90 MB was used to store the preprocessed aircraft performance data. `nF-iA*` reached 500 MB when the time limit was reached and `nF-A*` reached more than 5 GB in several instances.

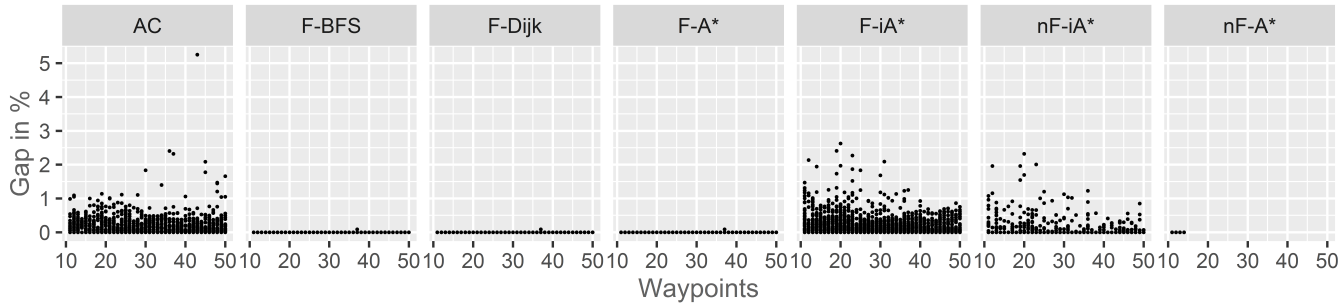

Figure 1. Scatter plot of percentage gap to the best solutions on each instance.

Cost assessment We ran each algorithm i on each instance j and recorded the cost returned by the algorithm, x_{ij}, and the cheapest cost found by any algorithm in our study on an instance, x^*_j. We then computed the percentage gap $y_{ij} = 100 \cdot (x_{ij} - x^*_j)/x^*j$. A scatter plot of this metric is reported in Fig. 1. We observe that `F-BFS`, `F-Dijk`, and `F-A`* find trajectories of exactly same cost. `nF-A`* also finds the same solutions in the instances it is able to solve. These solutions are also always the best except for one single instance where the two non-FIFO algorithms find a better solution. This is the only case out of 3600 where we observed that not assuming the FIFO property is necessary to achieve better solutions. For instances below 14 waypoints, where `nF-A`* terminated, a gap equal to zero indicates that the solution is provably optimal. For more waypoints the optimal solutions could be better than those found in our experiments and hence the impact of assuming FIFO might be more pronounced. Another explanation for the low impact of the FIFO assumption can be the low frequency of changes in our weather data. Given more frequent changes, the impact might be more relevant. It should also be noted that fluctuating weather conditions could make the optimal route bumpy, and, therefore, unpleasant for the passengers. Our algorithm could be adjusted to prevent bumps however this turned out to be unnecessary in our tests that resulted always in stable routes.

As expected, inconsistency and inadmissibility issues in both `F-iA`* and `nF-iA`* seem to have a considerable impact on the deterioration of solution quality. Finally, the solutions we found with any algorithm in this study are better than those of `AC`.

Computation cost assessment The running time of the whole trajectory optimization algorithm comprises the time to carry out preprocessing computations, where needed, and the time to execute findPath until convergence. The number of calls to findPath was in most of the cases two or three with peaks of four or five with many waypoints. There is no significant influence on this number by the algorithm used to implement findPath, hence in the analysis that follows we consider results only on preprocessing and single runs of findPath. A scatter plot of the number of opened and expanded nodes and of the computation time in milliseconds is reported in Fig. 2.

It is evident from the figure that the number of labels opened and expanded is much higher in non-FIFO algorithms. As a consequence several runs do not terminate within the time limit. For this reason we stopped prematurely running `nF-A`* on instances with more than 14 waypoints. The use of the inconsistent and inadmissible heuristic in `nF-iA`* was instead effective. It reduces by a good margin the amount of labels opened and expanded, and thus `nF-iA`* can solve also the instances with high number of waypoints, although still not all. This comes at the cost of suboptimal routes though.

We observe that `F-Dijk` opens fewer nodes than `F-BFS` but expands about the same number of nodes and hence has almost the same running times. Optimal paths typically climb to a high altitude and then cruise at that altitude for most of their lengths. However, `F-Dijk` gets trapped expanding labels of descending nodes, as these are almost always cheaper. Whenever `F-BFS` opens a label, it only needs to insert it into a list, whereas `F-Dijk` needs to add it to the red-black tree. This fact has however no evident consequence on the running time, which is anyway dominated by the calculation of the cost of an arc and, hence, by the number of labels expanded.

`F-A`* opens and expands less nodes than `F-Dijk` and is therefore also slightly faster, even including the preprocessing time required to do the backwards breadth-first search to find the heuristic values. However, as mentioned earlier the bound is not very strong and thus the decrease in search space is not impressive. `F-iA`* reduces the search space by a much larger margin, and is also considerably faster. Once again though, this comes at the cost of suboptimal routes.

6 Conclusions

We found impractical from a computational point of view finding optimal trajectories without assuming the FIFO property. Then, our best suboptimal algorithm without assuming FIFO is `nF-A`*. However, we found only a single instance out of 3600 tested, where this algorithm finds better solutions than those found by much faster algorithms that instead assume the FIFO property. To solve the problem in a FIFO setting, either `F-A`* or `F-iA`* is the best choice, depending on whether solution cost or running time is deemed most important. `F-A`* finds always the solution of best cost, which is optimal in the most common case that the FIFO property holds in the data. `F-A`* finds less good solutions but is faster.

The design of a heuristic for A* that makes the algorithm optimal while maintaining its efficiency is a possible focus for further research. However, important savings in costs could be found by an integrated solution approach for 3D route optimization, where the 2D route and the trajectory are optimized together.

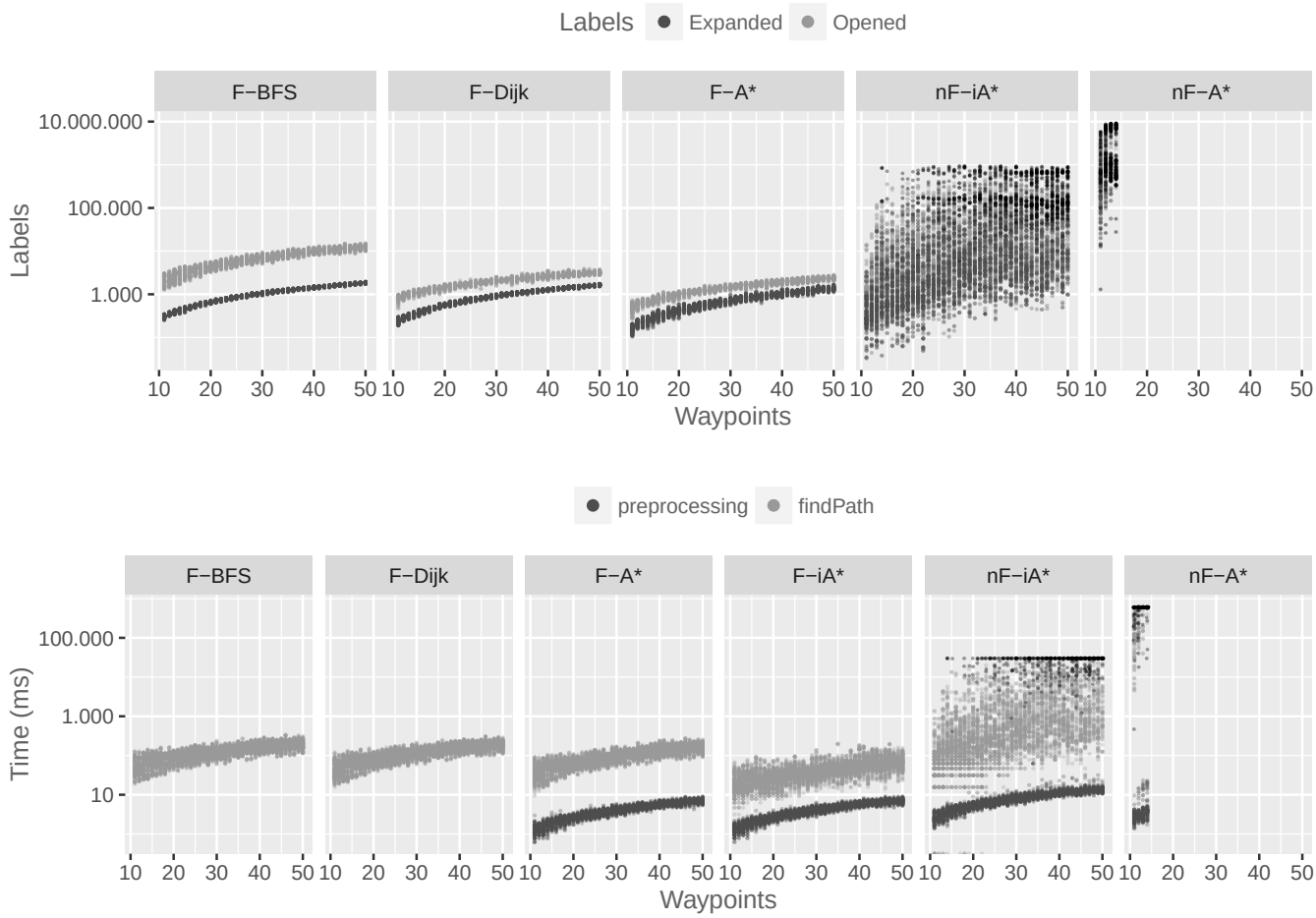

Figure 2. Above, semi-logarithmic plot of the number of opened and expanded nodes; below, semi-logarithmic plot of the computation time of preprocessing and findPath. There is no preprocessing in `F-BFS` and `F-Dijk`. Data points related to runs that exceeded the time limit are marked in black.

References

[1] H. Bast, D. Delling, A. Goldberg, M. Müller-Hannemann, T. Pajor, P. Sanders, D. Wagner, and R.F. Werneck, 'Route planning in transportation networks', Technical Report arXiv:1504.05140 [cs.DS], arXiv, (2015).

[2] G.V. Batz, R. Geisberger, P. Sanders, and C. Vetter, 'Minimum time-dependent travel times with contraction hierarchies', *ACM Journal of Experimental Algorithmics*, **18**(1), Article No. 1.4, (2013).

[3] R. Bellman, 'On a routing problem', Technical Report P-1000, Defense Technical Information Center, (1956).

[4] I. Chabini and S. Lan, 'Adaptations of the A* algorithm for the computation of fastest paths in deterministic discrete-time dynamic networks', *IEEE Transactions on Intelligent Transportation Systems*, **3**(1), 60–74, (2002).

[5] K.L. Cooke and E. Halsey, 'The shortest route through a network with time-dependent internodal transit times', *Journal of mathematical analysis and applications*, **14**(3), 493–498, (1966).

[6] D. Delling and G. Nannicini, 'Bidirectional core-based routing in dynamic time-dependent road networks', in *International Symposium on Algorithms and Computation (ISAAC)*, volume 5369 of *LNCS*, pp. 812–823. Springer, (2008).

[7] E. W. Dijkstra, 'A note on two problems in connexion with graphs', *Numerische Mathematik*, **1**(1), 269–271, (1959).

[8] P. E. Hart, N. J. Nilsson, and B. Raphael, 'A formal basis for the heuristic determination of minimum cost paths', *IEEE Transactions on Systems Science and Cybernetics*, **4**(2), 100–107, (July 1968).

[9] D.E. Kaufman and R.L Smith, 'Fastest paths in time-dependent networks for intelligent vehicle-highway systems application', *Journal of Intelligent Transportation Systems*, **1**(1), 1–11, (1993).

[10] C. Y. Lee, 'An algorithm for path connection and its applications, ec-10(3)', *IRE Transactions on Electronic Computers*, 346–365, (1961).

[11] Edward F. Moore, 'The shortest path through a maze', in *The International Symposium on the Theory of Switching*, pp. 285–285. Harvard University Press, (1959).

[12] A. Orda and R. Rom, 'Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length', *Journal of the ACM*, **37**(3), 607–625, (1990).

[13] R.S.F. Patrón, Y. Berrou, and R. Botez, 'Climb, cruise and descent 3d trajectory optimization algorithm for a flight management system', in *AIAA/3AF Aircraft Noise and Emissions Reduction Symposium*. American Institute of Aeronautics and Astronautics (AIAA), (June 2014).