Constraint Handling in Flight Planning

Anders N. Knudsen, Marco Chiarandini, and Kim S. Larsen*

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

Abstract. Flight routes are paths in a network, the nodes of which represent waypoints in a 3D space. A common approach to route planning is first to calculate a cheapest path in a 2D space, and then to optimize the flight cost in the third dimension. We focus on the problem of finding a cheapest path through a network describing the 2D projection of the 3D waypoints. In European airspaces, traffic flow is handled by heavily constraining the flight network. The constraints can have very diverse structures, among them a generalization of the forbidden pairs type. They invalidate the FIFO property, commonly assumed in shortest path problems. We formalize the problem and provide a framework for the description, representation and propagation of the constraints in path finding algorithms, best-first, and A* search. In addition, we study a lazy approach to deal with the constraints. We conduct an experimental evaluation based on real-life data and conclude that our techniques for constraint propagation work best together with an iterative search approach, in which only constraints that are violated in previously found routes are introduced in the constraint set before the search is restarted.

1 Introduction

The Flight Planning Problem (FPP) aims at finding 3D paths for an aircraft in an airway network, minimizing the total cost determined by fuel consumption and flying time. The motivation is financial and environmental. Airway networks can be huge, due to the added dimension compared with road networks, and side constraints complicate the problem further. 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 and to minimize the need for later changes by the institution itself. Therefore, the common practice is to determine the precise flight route only a few hours before take-off. For this to be feasible and bring any advantage, the route determination must be quite 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. Over the last years, the strain on the European airspace has increased to a level where the network must be heavily

* This author was supported in part by the Danish Council for Independent Research, Natural Sciences, grant DFF-1323-00247.

constrained to ensure safe flights. This also implies increased difficulty in finding cost-efficient routes respecting the constraints.

We focus on the problem of finding 2D routes in European airspaces in an off-line setting. Normally, waypoints are defined for one or more intervals of flight levels, but here we assume that flights are cruising at a given altitude. This version of the problem is relevant because a common approach to flight planning in industry is to decompose the problem into two subproblems: finding a 2D route and expanding it in the third dimension. For both problems, there are constraints to satisfy and costs to minimize. Moreover, costs are resource-dependent because they depend on the weather conditions, which vary with time, and on the weight of the aircraft. This latter depends on the fuel level, the initial amount of which is also a decision variable of the problem. We use an estimate based on a great circle distance for this initial amount of fuel and assume that its more precise determination is done during the vertical route optimization (see [1], for instance).

The classic shortest path problem has been the focus of considerable amounts of research for many years. For an extensive survey on recent advances, see [2]. However, many of the new advances rely on preprocessing techniques, most of which we deem inapplicable in the flight planning context, due to the impact of the constraints. The problem of finding cheapest flight routes with resource-dependent costs was studied in [3] and [4], and more recently in [5]. The latter focuses on a 2D version, presenting three A* algorithms with different heuristic functions. However, constraints are not taken into account in these works while they are the main focus of our work.

The constraints in the European airspaces come in three different forms: Conditional Routes (CDR), Route Availability Document (RAD), and Restricted Airspaces (RSA). These are all published by the European air traffic management institution, Eurocontrol. RAD constraints are the most general and challenging. They include local constraints affecting the availability of airways and airspaces at certain times, but they are primarily conditional types of constraints. For example, if the route comes from a given airway, then it can only 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 fly over certain airways. Or short-haul and long-haul are segregated in congested zones. RAD constraints must be handled during the route construction or checked later. There are more than 16,000 of these constraints and they can be updated several times a day, although most of them remain unchanged for longer time periods.

Some of these constraints are generalizations of the *forbidden pairs* type, which make the problem at least as hard as the *path avoiding forbidden pairs* problem that was shown to be NP-hard [6]. Given a topological sorting of the nodes, restricting to certain structures of forbidden pairs makes the problem polynomially solvable [7]. However, none of these structures can be guaranteed in the European airspace network.

Our contribution is the design of a framework for the representation and propagation of RAD constraints during the search. We formalize the constraints and extend path finding algorithms, such as best-first and A* search [8], to handle them. In particular, we propose an ad hoc tree structure to represent the constraints and to check their satisfiability and implications and to simplify their structure during the search. Then, we study two lazy approaches to constraint propagation. In one approach, we postpone the expansion of partial paths that cannot be dominated due to the constraints, but that are less promising than others in terms of costs. We then reconsider them only if it becomes necessary. In the other approach, which is similar to a Bender's decomposition with nogood cuts, we ignore all constraints in an initial search. If the path found is feasible, then we found a solution. Otherwise, we include only those constraints that are violated by the current path and iterate the whole process until a feasible solution is found. Additionally, we consider an exact and a heuristic approach to removing active constraints during the search based on geographical considerations.

This work is in collaboration with an industrial partner. Their core business is in flight route planning. Many of their customers are owners of private planes who plan their flights shortly before departure. Once they have chosen a destination, they send a query for a route from some portable device and they expect an almost immediate answer. Hence, this company is interested in an algorithm that can solve the problem within a few seconds. The size of the network used by this company is approximately 100,000 nodes and 3,000,000 edges and we use these real life data to test our ideas.

2 The Constrained Horizontal Flight Planning Problem

The European airspace is a network of *waypoints* that can be traversed at different altitudes (*flight levels*). Waypoints are connected across different flight levels by *airways*. The overall network could be described as a layered digraph, with several nodes for each waypoint representing different flying altitudes and arcs connecting these nodes if they belong to different waypoints. We simplify the situation by only allowing flights at a single flight level. The flight level is chosen to be the best cruising altitude for the tested aircraft. Hence, we represent the European airspace as a 2D network formed by a directed graph $D = (V, A)$, where the nodes in V represent waypoints defined by latitude and longitude coordinates and the arcs in A represent feasible airways between the waypoints. Each arc has associated resource consumptions and costs. The resource consumption for flying through 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 of the consumption, respectively. The cost c_a is a function of the resource consumption, i.e., $c_a = f(\boldsymbol{\tau}_a)$.[1] A *2D (flying) route* is an (s, g)-path in D represented by n waypoints plus a departure node (source) s and an arrival node (goal) g, that is, $P = (s, v_1, \ldots, v_n, g)$,

[1] The total cost is calculated as a weighted sum of time and fuel consumed. In our specific case, we have used 3$ per gallon of fuel and 1000$ per hour.

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 a route 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 the elements, on the time window, and on the flight level range (although the latter does not play a role in our 2D setting).

Definition 1 (Constrained Horizontal Flight Planning Problem). *Given a network $N = (V, A, \boldsymbol{\tau}, \boldsymbol{c})$, a departure node s, an arrival node g, and a set of side constraints $\mathcal{C}$, find an (s, g)-path P in D that satisfies all constraints in $\mathcal{C}$ and that minimizes the total cost, c_P.*

We use the abbreviation CHFPP for the above.

In most common shortest path problems, a property that usually holds is the First In First Out (FIFO) property. It states that a path P' reaching a node with a cost worse than another path P reaching the same node cannot become part of the final solution and can, therefore, be discarded. This property plays a fundamental role in the efficiency of both Dijkstra and A* algorithms.

However, this property does not hold in our CHFPP. Indeed, a path P' arriving at a node with a cost worse than another path P reaching the same node cannot be discarded, because if the conditions activated during the path P' are less stringent than those activated during the path P, then P' could still become part of the best route. Moreover, the performance on a given arc is influenced by the weight (which in turn depends on the fuel consumed up to that arc) and by the time at which the arc is traversed (due to the possibly changing weather conditions). These dependencies of the resource consumptions on the path up to a given point are reflected in the cost of the next arcs, which, therefore, cannot be statically determined. Therefore, because of these dependencies of the cost function, the FIFO property would not hold even if there were no constraints. However, our experiments (see Section 4) show that for the real cases studied, no optimal solution is missed by assuming the FIFO property on cost. Hence, to simplify the presentation, we will assume the FIFO property on cost, but emphasize that we will *not* assume the FIFO property on constraints.

2.1 Definition of RAD Constraints

RAD constraints are implications of two types: *forbidden* and *mandatory*. They consist of an *antecedent* expression p and a *consequent* expression q. The expressions are Boolean and contain identifiers of locations visited during the flight and relationships between these. A RAD constraint is *satisfied* when the antecedent is false or when an antecedent is true and the consequent is true (in the mandatory case) or false (in the forbidden case). Thus, the interpretation is $p \rightarrow q$ for the mandatory and $p \rightarrow \neg q$ for the forbidden case. On the other hand, a RAD constraint is *violated* if it is mandatory and $p \rightarrow q$ is false or if it is forbidden

```
constraint : 'Forbidden:' ID 'Antecedent:' expr 'Consequent:' expr
           | 'Mandatory:' ID 'Antecedent:' expr 'Consequent:' expr
expr_list  : expr
           | expr expr_list
expr       : '(' AND expr expr_list ')'
           | '(' OR expr expr_list ')'
           | '(' SEQ expr expr_list ')'
           | '(' NOT expr ')'
           | term
           | term time
term       : point | airway | airspace | arrival | departure
point      : 'Point:' ID
           | 'Point:' ID 'FL:' FLIGHT_LEVELS
airway     : 'Airway: from' ID 'to' ID
           | 'Airway: from' ID 'to' ID 'FL:' FLIGHT_LEVELS
airspace   : 'Airspace:' ID 'FL' FLIGHT_LEVELS
departure  : 'Dep:' ID
arrival    : 'Arr:' ID
time       : 'Time:' date 'to' date '-' time 'to' time '-' WEEKDAYS
```

Fig. 1. Bison (yacc) grammar for RAD constraints

```
Forbidden: ID xxxx
Antecedent: (AND (OR Airspace: eg Airspace: ee) (NOT
     Point: mohni))
Consequent: Point: petot FL: 0-200
Time: 03-07-16 to 20-12-16 - 08:00 to 16:00 - FrSaSu
```

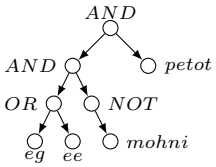

Fig. 2. Example of a forbidden constraint and its tree representation.

and $p \rightarrow \neg q$ is false. In Fig. 1, we have defined a grammar to specify all possible types of RAD constraints.

Expressions are written in prefix notation using non-binary operators. Besides well-known operators, there is `SEQ` (sequence) for which all operands must be satisfied in the same order as they are presented in the constraint. The terms represent the possible flight choices, such as waypoint, airways between them, airspaces, and departure/arrival airports. `ID`'s are the identifiers of the respective terms. Note that flight levels, included in the grammar for completeness, are not relevant in this exposition. Terms that have a `time` associated with them, are only satisfied if they are visited within the specified time window. An example of a constraint can be seen to the left in Fig. 2.

3 Path Finding Algorithms

In this section, we present path finding algorithms for solving the CHFPP to optimality. We consider classic best-first and A* search modified to take the constraints and the cost dependencies into account. Then, we introduce lazy approaches to deal with constraints, both during the search and after the search, leading to an iterated search process.

3.1 Handling the Constraints

In our path finding algorithms, constraints are checked while the route is constructed. Each RAD constraint is encoded in a tree data structure, where leaves are terms and internal nodes are operators (see Fig. 2, right). The truth values of the leaves are propagated up to the root, which must evaluate to false for the route to be feasible with respect to the corresponding constraint. Initially, all terms are in an unknown state. Then, if a term is resolved, the respective term is removed from the tree and the truth value is propagated upwards.

The set of constraints $\mathcal{C}$ is translated 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.

Partial paths under construction are represented by labels. A *label* ℓ is associated with a node $\phi(\ell) = u \in V$ and contains information about a partial route from the departure node $s \in V$ to the node u. It is written as $\ell = (P_\ell, c_\ell, \Delta_\ell)$, where $P_\ell = (s, ..., u)$ is the path taken, c_ℓ is the cost of the path, and Δ_ℓ is the set of constraint trees of active constraints for the label ℓ. *Active constraints* are those where at least one term in the antecedent or consequent part has been determined, but where the complete satisfaction of the constraint has not yet been decided. Note that the constraint trees in Δ_ℓ are different from the initial ones in Γ because some terms may have been resolved and the tree consequently reduced. Formally, $\Delta_\ell = \{\rho(\gamma, P_\ell) \mid \iota(\gamma) \in E_u \cup E_{uv}, uv \in P_\ell\}$, where $\rho(\gamma, P_\ell)$ is the tree $T(\gamma)$ after propagation of the terms in P_ℓ. However, active constraints preserve the original identifiers, that is, $I(\Delta_\ell) = \{\iota(\gamma) \mid \gamma \in \Delta_\ell\} = \{\iota(\gamma) \in E_u \cup E_{uv}, uv \in P_\ell\}$. Depending on whether the term is negated or not, some locations can be advantageous or disadvantageous for a label to visit, opening up or restricting possibilities ahead This can be determined for each term while building the constraints and active constraints are flagged as belonging to one of the two categories when a term is resolved.

All labels created are maintained in a structure $\mathcal{Q}$, called the *open list*. The *expansion* of a label is the operation of extracting a label from $\mathcal{Q}$ and inserting a new label into $\mathcal{Q}$ for any node in D reachable by an outgoing arc from the node of the label under expansion. When a label ℓ with $\phi(\ell) = u \in V$ is expanded along an arc $uv \in A$, a new label $\ell' = ((s, \ldots, u, v), c_\ell + c_{uv}, \Delta_{\ell'})$ is created. The new set of constraint trees 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 satisfaction of 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 route would be infeasible. 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

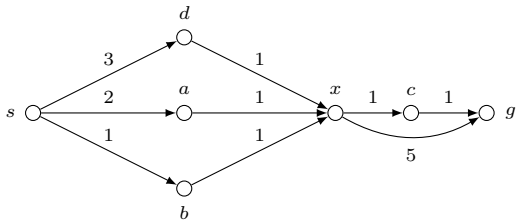

Fig. 3. An example where a partly dominated label leads to an optimal route. The labels are $\ell_1 = ((s, a, x), 3, \{C_1\})$, $\ell_2 = ((s, d, x), 4, \varnothing)$ and $\ell_3 = ((s, b, x), 2, \{C_1\})$.

deduced appears in the path. Formally, $\Delta_{\ell'} = \{\rho(\gamma, P_{\ell'}) \mid \gamma \in \Delta_\ell\} \cup \{\rho(\gamma, P_{\ell'}) \mid \iota(\gamma) \in E_{\phi(\ell')} \cup E_{\phi(\ell)\phi(\ell')}\}$.

For efficiency reasons, we use the following conservative approximation of logical implication between sets of constraints. We say that Δ_{ℓ_a} is *implied* by Δ_{ℓ_b} if no constraints in $I(\Delta_{\ell_b}) \setminus I(\Delta_{\ell_a})$ are marked as advantageous, no constraints in $I(\Delta_{\ell_a}) \setminus I(\Delta_{\ell_b})$ are marked as disadvantageous, and the trees of all constraints in $I(\Delta_{\ell_a}) \cap I(\Delta_{\ell_b})$ are identical, in the sense that they are isomorphic when regarded as ordered trees (sorted in the order of the input of the corresponding constraints).[2] Further, we hash a post-order traversal of the tree so that identity check is fast. The traversal is performed anew any time the tree is evaluated (during an expansion) and the hash function is recomputed and stored at the same time. If we hash to a 64 bit value, false positives are extremely unlikely.

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 the open list and deleted. 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 *partly dominated* by ℓ_b. Partly dominated labels cannot be removed from the open list. As an example, consider the scenario in Fig. 3. Let $C_1 = ((a \vee b) \wedge c)$ be the only constraint relevant to the example. 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. The label ℓ_1 is dominated by ℓ_3 and can be discarded. The route is cheaper and Δ_{ℓ_3} is implied by Δ_{ℓ_1}, as they both contain only C_1. On the other hand, ℓ_2 is only partly dominated by ℓ_3, because Δ_{ℓ_3} is not implied by Δ_{ℓ_2}. Hence, ℓ_2 is not discarded. Indeed, ℓ_2 leads to the cheapest route to g, since ℓ_3 must avoid c while ℓ_2 does not have to.

3.2 Best-first and A* Algorithms

Our algorithms are based on classic best-first and A* algorithms. These algorithms expand labels from an open list $\mathcal{Q}$ until a path from source to goal is proven optimal. When we extract a label from the open list, we choose one of

[2] Note that a more accurate determination of subsumption between two trees, accurately reflecting semantic logical implication, would require solving a subgraph isomorphism that can be quite costly due to its NP-completeness.

smallest cost (best-first) or smallest sum of the cost of the label and a heuristic estimate of the cost from the corresponding node to the goal (A*). The algorithm terminates when the goal g has been reached and the incumbent best path to g is cheaper than the cheapest label in $\mathcal{Q}$. In best-first, the solution returned is optimal. In A*, the solution returned is optimal if the heuristic is both admissible (the estimated cost must never overestimate the cost from a node to the goal) and consistent (for every node u, the estimated cost of reaching the goal must not be greater than the cost of getting to a successor v plus the estimated cost of reaching the goal from v). Consistency can be shown to be a stronger property as it also implies admissibility. As heuristic, we use the cheapest path determined by preprocessing the graph with a backward breadth-first search from the goal to all other nodes. The guarantee of admissibility and consistency of these estimates is obtained by disregarding the constraints and assuming a cost on each arc that is a lower bound of the corresponding costs. The lower bound can be computed by choosing the best weather conditions in the period between the departure time and an upper bound on the arrival time.

A baseline of the resulting path finding algorithm for solving CHFPP is given in Alg. 1. The function FindPath takes the initial conditions of the aircraft as input, i.e., the initial fuel load τ_0^x, the departure time τ_0^t, a network $N = (D, \boldsymbol{\tau}, \boldsymbol{c})$ built using information from the airspace, aircraft performance data, and weather conditions. Here, $\boldsymbol{\tau}$ and $\boldsymbol{c}$ are intended as data tables. The time and fuel consumption for an arc is looked up in these tables using the inputs: (i) the fixed flight level, (ii) weight, (iii) international standard atmosphere deviation (i.e., temperature), (iv) wind component, and (v) cost index.[3] Inputs (ii), (iii), and (iv) depend on the partial path.

Differently from classic path finding algorithms, the algorithm in Alg. 1 includes an extra comparison with respect to constraints for the domination of labels (lines 20–21 and 23). Under the FIFO assumption, it would be possible to determine a strict domination among labels and to add nodes of expanded labels to a closed list. As a consequence, at most one label per node would be expanded. However, in our case, domination of labels also needs to take constraints into account, for which the FIFO property does not hold. Thus, partly dominated labels cannot be discarded and the closed list becomes unnecessary. As a consequence, more than one label from a node can be expanded.

Finally, although D contains cycles and although, theoretically, the cycles could be profitable because of the time dependency of costs, labels are not allowed to expand to already visited vertices because routes with cycles would be impractical.

[3] The cost index is an efficiency ratio between the time-related cost and the fuel cost that airlines use to specify how to operate the aircraft, determining the speed of the aircraft. It is decided upon at a strategic level and cannot be changed during the planning phase.

```
1  Function FindPath((τ_0^x, τ_0^t),N = (D(V, A), τ, c),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 ← extract the cheapest label from Q
6          if (c_ℓ > c_ℓr) then break                          ▷ termination criterion
7          if (φ(ℓ) = g) and (c_ℓ < c_ℓr) 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_ℓr and c_ℓr
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: A general template for solving CHFPP

3.3 Lazy Expansion

In Alg. 1, partly dominated labels are also added to $\mathcal{Q}$, so only few labels can actually be dominated. To speed up the algorithm, we attempt a *lazy* approach to expansions by postponing the expansion of partly dominated labels. This is achieved as follows. At each node $v \in V$, we maintain a waiting list of labels, ω_v. Then, instead of adding partly dominated labels at a node v to $\mathcal{Q}$, we add them to ω_v. The idea is that if all successors of the cheapest label at v, $\ell = ((s, ..., v), c_\ell, \Delta_\ell)$, are able to expand throughout the cheapest path from ϕ_ℓ to g without being affected by constraints in Δ_ℓ, then there is no label that was partly dominated by ℓ that would lead to a better route. However, if there is a successor ℓ' of ℓ that cannot expand to the next node in the cheapest path from ϕ_ℓ, then one of the labels in ω_v could potentially lead to a better route, and thus must be inserted into $\mathcal{Q}$. This is done by *backtracking* through every node in the path of the label ℓ' and, at each node in $P_{\ell'}$, inserting into $\mathcal{Q}$ the cheapest label from the corresponding waiting list.

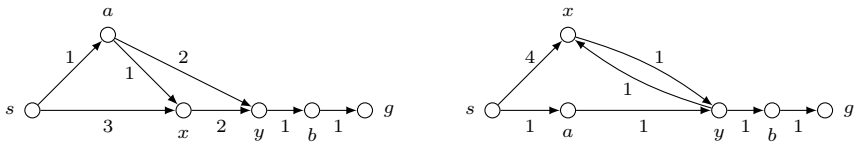

Fig. 4. Backtracking triggered by domination (left) and cycling (right)

Backtracking is triggered whenever a label cannot be expanded to a reachable node because a constraint becomes violated and the path infeasible or whenever a label is dominated. An example of backtracking due to infeasibility was presented in Fig. 3. There the label l_2 is partly dominated by l_3 and hence set in ω_x, but when l_3 fails to expand to c, l_3 is backtracked, resuming l_2, which is moved from ω_x to $\mathcal{Q}$. For an example where backtracking is needed because of domination, consider the situation in Fig. 4 (left). The only relevant constraint is $C_2 = (a \wedge b)$. At the node x, we have the labels: $\ell_1 = ((s, a, x), 2, \{C_2\})$ and $\ell_2 = ((s, x), 3, \varnothing)$ with ℓ_2 in ω_x because of being partly dominated by ℓ_1. At the node y, the labels are: $\ell_3 = ((s, a, x, y), 4, \{C_1\})$ (which is the expansion of ℓ_1) and $\ell_4 = ((s, a, y), 3, \{C_1\})$. When the label ℓ_3 is discovered to be dominated by ℓ_4, it cannot simply be removed because then we would lose the label ℓ_2 that, when expanded to y, becomes $\ell_5 = ((s, x, y), 5, \varnothing)$, which is only partly dominated by ℓ_4. Hence, we need to backtrack ℓ_3 and include ℓ_2 in $\mathcal{Q}$.

When we backtrack a label ℓ to a given node u, we select the cheapest label ℓ' from the waiting list at u and add that to the open list. We only need to backtrack ℓ once, since backtracking ℓ' will trigger further moves to the open list, if it becomes necessary. Therefore, we associate a backtracking indicator with each label to prevent backtracking from the same label a second time.

Particular care must be devoted to potential cycles. Routes are not allowed to visit the same node twice, so the detection of cycling in D can also be the cause of a label not being expanded. Consider the situation in Fig. 4 (right). The only relevant constraint is $C_3 = (a \wedge b)$. The labels at x are $\ell_1 = ((s, a, y, x), 3, \{C_3\})$ and $\ell_2 = ((s, x), 4, \varnothing)$, with ℓ_2 in ω_x because of being partly dominated by ℓ_1. Further, at b, we have the label $\ell_3 = ((s, a, y, b), 3, \{C_3\})$. When we try to expand ℓ_1, we discover it cannot be expanded anywhere without creating a cycle. Then we consider ℓ_3, and discover that it has become infeasible. However, backtracking ℓ_3 does not allow us to resume ℓ_2 from ω_x because we do not pass through x. Thus, ℓ_2 would never be added to $\mathcal{Q}$ and we would not find the one feasible route. Hence, backtracking must be triggered also when cycles are detected.

To handle this efficiently, we equip all labels ℓ with a dictionary, H, associating nodes with labels. The keys of such a dictionary are the nodes of the path P_ℓ, and the associated value, $H(u)$, is the label at u which is eventually expanded into ℓ. We use a small hash table and get expected constant time look-ups. After the initialization of the dictionary, cycles can be detected in constant time by a look-up. Additionally, we let $\pi(\ell)$ denote the label associated with the predecessor of the last node in P_ℓ.

Further, it should be noted that, when backtracking is caused by domination or cycle detection, it can be delayed. Let ℓ' be a label that we need to backtrack and let ℓ be the *blocking* label, that is either the dominating label (domination case) or $H(u)$ if ℓ' is trying to expand to a node u that is already in $P_{\ell'}$. Let $B_{\ell'}$ denote the set of labels that would be added to $\mathcal{Q}$, if ℓ' were to be backtracked immediately. Since the labels in $B_{\ell'}$ were all partly dominated predecessors of ℓ', any successor of those reaching $\phi(\ell')$ would be more expensive than ℓ' and thus they would be (partly) dominated by ℓ as well. Therefore, backtracking can be delayed until ℓ is backtracked, which would allow the successors of labels in $B_{\ell'}$ to reach farther than $\phi(\ell)$.

To implement delayed backtracking, we add to the information maintained with each label ℓ a list β_ℓ of labels that were blocked by ℓ at some point. Thus, whenever a label ℓ' is blocked by ℓ and should be backtracked, we do the following. If ℓ has already been backtracked, we backtrack ℓ' immediately, but otherwise, we delay and add ℓ' to β_ℓ instead. When ℓ itself is backtracked, besides $\pi(\ell)$, also all labels in β_ℓ are backtracked.

Theorem 1. *Algorithm 1 with lazy expansion returns optimal routes.*

Proof. The algorithm is derived from Algorithm 1, which is optimal, by adding lazy expansion. To show that lazy expansion maintains optimality, we need to show that all labels that are still in any waiting list when the algorithm terminates cannot be part of an optimal (s, g)-path.

Let ℓ be a label at $v \in V$, stored in ω_v when the algorithm terminates. Since ℓ is in ω_v, there must exist a label ℓ' which partly dominated ℓ, i.e., $c_{\ell'} < c_\ell$. Since ℓ is still in ω_v when the algorithm terminates, none of the expanded successors of ℓ' can have caused a backtrack. Thus, any possible path from v to any node in V originating from ℓ has also been explored by the expanded successors of ℓ' and is also cheaper.

3.4 Further Elements: Lazy Constraints and Constraint Pruning

Lazy Constraints and Iterative Path Finding. An alternative approach is to ignore the constraints initially and to iterate the search, adding constraints only when they are actually violated in the route found. First, a route is found without considering any RAD constraints. Then, the route is checked against all constraints. If no constraints are violated, the route is valid and the algorithm terminates. Otherwise, if one or more constraints are violated, the constraints are added to the input data of the path finding algorithm and a new search is started. The advantage of this procedure is that it avoids considering many constraints that never turn out to be relevant for the optimal route.

Heuristic Constraint Pruning. Some active constraints may become very unlikely to be violated if their terms correspond to locations that are already passed by the label or far from the direct route between the current node of the label and the goal. Thus, whenever during expansion a label ℓ evaluates a constraint, we

try to estimate heuristically whether it is still relevant or not. More precisely, we compare a lower bound and an upper bound for the length of a route from $\phi(\ell) \in V$ to the destination passing through the location $u \in V$ (or $uv \in A$) represented by the term. If the lower bound is larger than the upper bound, then we declare the term not satisfiable. Let $d(P)$ be the flying distance covered by a path P and $\gcd(u, v)$ be the great circle distance between two airway points u and v. The lower bound is given by $d(P_\ell) + \gcd(\phi(\ell), u) + \gcd(u, g)$. We use two different heuristic values for the upper bound. One is the *current result*: once the search finds any feasible (s, g)-path, with ℓ' being the final label, $d(P_{\ell'})$ is saved as the upper bound. If a better (s, g)-path is found, the bound is updated. The second heuristic is the *remaining distance*. It uses $d(P_\ell) + (1.3 \cdot \gcd(\phi(\ell), g))$ as the upper bound. The factor 1.3 was determined by observing the maximal deviation of historical routes from the great circle distance. This heuristic is disabled when close (i.e., within 20 nautical miles) to the departure or destination, as the procedures to exit and enter airports are unpredictable and can deviate considerably from great circle distances.

4 Experimental Results

Experimentally, we have compared different algorithms obtained from the combination of the elements presented in the previous sections. We consider computation time, number of labels expanded, and the quality of the routes.

We use real-life data provided by our industrial partner. This data consists of aircraft performance data, weather forecast data in standardized GRIB2 format, and a navigation database containing all the information for the graph. The graph consists of approximately 100,000 nodes and 3,000,000 edges. The aircraft performance data refers to one single aircraft and tests are run on the optimal cruising flight level of that aircraft, i.e., the one that yields the best cost on average weather conditions. The data for the weather forecast is given at intervals of three hours on specific grid points that may differ from the airspace waypoints. They are then interpolated both in space and time. 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 pursue a uniform coverage of the constraints in the network. Among the 156 possible pairings, 14 were discarded because of short distance, resulting in 142 pairs that were used as queries. Great circle distances range from 317 to 1721 nautical miles. All algorithms were implemented in C# and the tests were conducted on a virtual machine in a cloud environment with an Intel Xeon E5-2673 processor at 2.40Ghz and with 7GB RAM. To account for fluctuations in CPU time measurement, each algorithm was run 5 times on each instance and only the fastest was recorded. A preliminary comparison between A* and best-first unveiled that best-first is impracticable. Within a time limit of one minute, it terminates only in 11 instances against 103 of A*.

Assessment of the FIFO assumption on costs. We tested whether assuming the FIFO property on costs would lead to suboptimal results. Removing the

FIFO property means that labels in the open list are never dominated. More specifically, we tested two versions of A*, one that assumes FIFO, and thus is as described in the previous sections, and one where the lines 18–23 of the INSERT function in Alg. 1 are omitted. With a timeout of 10 minutes, A* without the FIFO assumption solved 78 out of 142 instances, and in these instances, all returned solutions were of the same cost as A*. Thus, we conclude that at least for our real-life setting, assuming the FIFO property on costs seems to be a good heuristic that does not affect the optimality of results. Henceforth, we continue to assess only algorithms that use this assumption.

Empirical run-time. We include in the run-time of the path finding algorithms both the time used for preprocessing (determining lower bounds for each vertex) and the time spent for actually performing the search. Initially, we compare the run-time time of 4 different algorithms: A*, A* with the upper bound heuristic to ignore constraints (A*UB), A* with the remaining distance heuristic (A*RD), and iterative A* (iA*). We use A* as a baseline algorithm and calculate the percentage deviation of running time per instance of the other algorithms with respect to A*. A scatter plot of the run-time percentage deviations from A* is shown in Fig. 5 (left column), where the x-axis represents different instances sorted by great circle distance between query airports. A time limit of one minute was used in these experiments. Within this time limit, A* did not terminate in 39 queries. These cases are detectable by the lack of points for some ordinate in the first panel in Fig. 5. There does not seem to be a correlation between the size of the instance and the non-termination of A*.

We observe that A*UB keeps returning optimal solutions (not shown), only results in minor runtime improvements compared to A*, and does not terminate in the same 39 instances. Separately, we observed that A* does little work after finding the first path to the goal, indicating that the heuristic cost value used for selection in all our A* algorithms must be very close to the exact value. Thus, since A*UB has an impact only after an (s, g)-path has been found, the space for improvement is small.

A*RD is considerably faster than A* and the number of instances unsolved within 1 minute is reduced to 15. Unfortunately, the omission of constraints is sometimes too optimistic, leading to suboptimal routes due to the inaccurate domination of some labels. This happens in 11 out of the 142 instances where the solution quality was within $0.1 - 0.6\%$ of the optimal solution. This effect can be controlled by increasing the 1.3 factor in the remaining distance heuristic, but this increases the running time. On the other hand, we never experienced that A*RD returned infeasible solutions (which could theoretically happen).

The winner of the comparison is by far iA*. The reduction in computation time with respect to A* is up to 99% in all instances going from running times of the order of seconds to running times of the order of milliseconds. It solves all cases where A* does not terminate, taking 12 seconds in the worst case (which is an extreme outlier in iA* running times). The number of iterations ranges between 1 and 10 and although the overall number of expanded labels can in

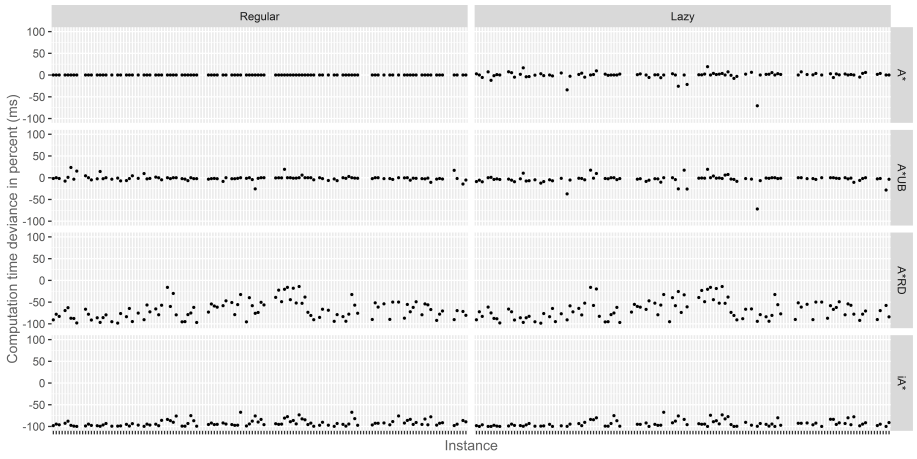

Fig. 5. Regular algorithms (left) and lazy expansion (right).

some cases become comparable to that of A*, the reduction in computation time from not having to handle a large number of constraints is huge.

In the right column of Fig. 5 we assess the impact of the lazy expansion to all four algorithms. The deviations are still calculated with respect to the results of the baseline A* algorithm. The visual comparisons performed row-wise inform us that the the lazy expansion improves the running time of the algorithms only in few cases. While in many instances there is a reduced number of label expansions, the overhead in run-time due to maintaining the waiting and backtracking lists is sometimes larger than the time saved.

Instance complexity. In Fig. 6, we investigate the scaling of the algorithms with respect to instance size. We removed iA* from the analysis because its running times and number of constraints activated are too small (note that iA* is however using A* and hence it is implicitly represented). The plots are on a semi-logarithmic scale with the run-time expressed in milliseconds on the y-axis. In the left column, we show the dependency on the great circle distance between the departure and destination airports of the query. We observe that there is no pattern in the points, indicating that this distance is not a good predictor for the complexity of the search. In the right column, we show the dependency of the run-time on the number of constraints that became active during the search. The plots indicate that this can be an important regressor hinting at an exponential relationship. Unfortunately, the number of constraints that become active is known only after the search has taken place. An analysis of the correlation between great circle distance and number of constraints was found to be inconclusive, hinting at the fact that it is not the length of the route but rather the density of constraints in the area it crosses that is important.

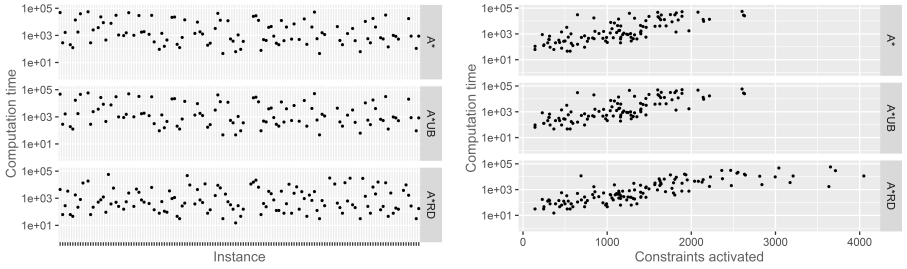

Fig. 6. Time complexity of the search as a function of distance (left) and as a function of constraints activated (right). The search is truncated at a time limit of one minute.

5 Conclusions

We have studied constraint handling in path finding algorithms for 2D route planning. We formalized the structure of these constraints and represented them with an ad hoc tree structure that makes it efficient to gradually update constraints and eliminate terms that become irrelevant during the search. We showed that from a collection of 16,000 constraints arising in a real-life setting, up to 4,000 were activated during the search of the algorithms. We concluded that a combination of constraint handling during the search and iterating A^*, introducing only relevant constraints, leads to significantly better running times than including all constraints from the beginning. We regarded this approach as a lazy constraint approach, but it can also be seen as a form of logic-based Benders decomposition driven by nogood cuts [9]. In our experiments, this approach reduced the running time of A^* from a few seconds to a few milliseconds. We also investigated another type of lazy approach, where the label expansions in path finding algorithms is conducted lazily. However, our experimental evaluation indicated that in our specific real-life instances, the contribution of this technique is not as pronounced as the lazy constraint approach. The handling of constraints during the search was new for our industrial partner, who decided to implement our algorithms in their product, obtaining an increased robustness and considerable reductions in running times.

We have also approached the problem with a generic purpose solver via mixed integer programming. If costs are considered static, the model is a classic min cost flow model with additional constraints derived from the RAD constraints that break the total unimodular structure of the constraint matrix. Preliminary results showed that this approach is slow. The instances were solved on average in about 12 minutes on an 8 core machine using about 10 GB of memory. However, this approach cannot deal with the—here fundamental—resource dependency structure of the costs. We expect this to be an issue with SAT solvers as well.

Throughout we have assumed a static flight level chosen as the one with best average performance. As future work, we plan to include the vertical dimension in our flight planning. The size of the network grows dramatically, and this leads to entirely new challenges.

References

1. Anders N. Knudsen, Marco Chiarandini, and Kim S. Larsen. Vertical optimization of resource dependent flight paths. In *Twentysecond European Conference on Artificial Intelligence (ECAI)*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, pages 639–645. IOS Press, 2016.
2. 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 arXiv:1504.05140 [cs.DS], arXiv, 2015.
3. Alberto Olivares, Manuel Soler, and Ernesto Staffetti. Multiphase mixed-integer optimal control applied to 4D trajectory planning in air traffic management. In *Proceedings of the 3rd International Conference on Application and Theory of Automation in Command and Control Systems (ATACCS)*, pages 85–94. ACM, 2013.
4. Hendrikus M. de Jong. *Optimal track selection and 3-dimensional flight planning: theory and practice of the optimization problem in air navigation under space-time varying meteorological conditions.* Staatsuitgeverij, 1974.
5. 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)*, pages 12:1–12:15. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2016.
6. Hananya Yinnone. On paths avoiding forbidden pairs of vertices in a graph. *Discrete Applied Mathematics*, 74(1):85–92, 1997.
7. Jakub Kováč. Complexity of the path avoiding forbidden pairs problem revisited. *Discrete Applied Mathematics*, 161(10–11):1506–1512, 2013.
8. Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.
9. John N. Hooker and Greger Ottosson. Logic-based benders decomposition. *Mathematical Programming*, 96(1):33–60, 2003.