Efficient Route Planning in Flight Networks*

Daniel Delling[1], Thomas Pajor[1], Dorothea Wagner[1], and Christos Zaroliagis[2,3]

[1] Department of Computer Science, Universität Karlsruhe (TH), P.O. Box 6980, 76128 Karlsruhe, Germany.
{delling,pajor,wagner}@informatik.uni-karlsruhe.de
[2] R.A. Computer Technology Institute, N. Kazantzaki Str., Patras University Campus, 26504 Patras, Greece
[3] Department of Computer Engineering and Informatics, University of Patras, 26500 Patras, Greece. zaro@ceid.upatras.gr

Abstract. We present a set of three new time-dependent models with increasing flexibility for realistic route planning in flight networks. By these means, we obtain small graph sizes while modeling airport procedures in a realistic way. With these graphs, we are able to efficiently compute a set of best connections with multiple criteria over a full day. It even turns out that due to the very limited graph sizes it is feasible to precompute full distance tables between all airports. As a result, best connections can be retrieved in a few microseconds on real world data.

Keywords: timetable information, flight modeling, shortest paths, multi criteria, table lookups

1 Introduction

Computing best connections in transportation networks is a showpiece application of algorithm engineering. The problem can be solved by modeling a transportation network as a graph where edge weights depict travel times on the corresponding connection. In general, DIJKSTRA's algorithm [11] can now solve the problem of finding the quickest path between two nodes s and t. One crucial challenge in the success of such an approach is the appropriate modeling of the transportation network as a graph. While road networks can be modeled in a straightforward manner (junctions are nodes, streets are edges), realistic modeling of public transportation networks is more complex [18, 22, 9].

A practical extension of the shortest path problem is route planning in a multi-modal context [15, 2, 1], where you switch—under certain constraints—the type of transportation during your journey. In this work, we deal with a subproblem of multi-modal route planning: Efficient computation of routes in flight networks. Our work is motivated from [10], where most of the time for

* Partially supported by the Future and Emerging Technologies Unit of EC, under contracts no. FP6-021235-2 (FP6 IST/FET Open/Project ARRIVAL) and no. ICT-215270 (FP7 ICT/FET Proactive/Project FRONTS), and the DFG (project WA 654/16-1).

J. Clausen, G. Di Stefano (Eds): ATMOS 2009
9th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems
http://drops.dagstuhl.de/opus/volltexte/2009/2145

retrieving best multi-modal connections is spent in a flight network, although the flight network makes up only a very small part of the whole (multi-modal) transportation network.

Related Work. Timetable information in flight networks is similar to railway networks, both inputs rely on some kind of periodic timetable. In addition, the best connection depends on the time of departure. Efficient models for route planning (or timetable information) in railway networks can be found in [18, 22, 9]. However, it turns out that simply using these models for flight networks yield unnecessary big graphs. To our best knowledge, no efficient model tailored to flight networks has been introduced yet.

In public transportation networks, we are not only interested in the best connection for a given departure time: we might be willing to alter our departure time in order to minimize the overall travel time. Such routes can be retrieved by profile-queries, where we compute all best connections for a full time period. Such profiles can be computed by a generalized variant of DIJKSTRA's algorithm [7] that propagates functions instead of scalars through the graph. An efficient algorithm for accelerating such queries in time-dependent road networks has been introduced in [8].

Moreover, the quickest connection is often not the best one: we also want to reduce the number of transfers and/or the costs of a journey. A possible approach to this is to compute a Pareto-set of routes [17, 12]. A route belongs to the Pareto-set if no other route is better or equal in all metrics (travel time, costs, transfers, etc.) under consideration. Pareto-routes can also be computed by a generalization of DIJKSTRA's algorithm [13, 14].

Our Contributions. In this work, we show how to plan routes in flight networks efficiently. Therefore, we first settle basic definitions on graphs and timetables in Section 2. Section 3 includes one of the main contributions of our work: flexible and yet compact time-dependent models tailored to route planning in flight networks. The key observation here is that in contrast to railway networks, flight networks contain (almost) only direct connections between airports. Unlike trains, planes do not stop at many airports on a route. Hence, we may use a different model yielding very small graphs.

In Section 4, we show how to retrieve best connections in flight networks. On the one hand, we deal with retrieving *all* quickest connections during the given time period, while on the other hand, we introduce two other metrics, i.e., transfers and travel costs, worth optimizing. We end up in a multi-criteria setup where the best connections form a Pareto-set. A key observation here is that the graphs deriving from our compact model are so small that we may afford to compute full Pareto-route distance tables between all pairs of airports in a preprocessing step. Then, queries are reduced to table-lookups yielding query times of a few microseconds.

In an extensive experimental study (Section 5), we show that our approach is indeed feasible for a real-world network consisting of roughly 1 000 airports.

Our constructed graphs are small, and computation of a Pareto distance table can be done in less than six minutes yielding a reasonable space consumption. With these tables at hand, queries can be accelerated by 5 orders of magnitude compared to a classic approach based on a multi-criteria DIJKSTRA. We conclude our work in Section 6 by a summary and possible future research.

2 Preliminaries

A *graph* is a tuple $G = (V, E)$ consisting of a finite set V of *nodes* and a set $E \subseteq V \times V$ of *edges* which are ordered pairs (u, v) if the graph is *directed*. The node u is called the *tail* of the edge, v the *head*. The reverse graph $\overleftarrow{G} = (V, \overleftarrow{E})$ is the graph obtained from G by substituting each $(u, v) \in E$ by (v, u).

Routing in public transportation networks requires an underlying timetable. In this work we restrict ourselves to periodic timetables with a fixed time period $\Pi \in \mathbb{N}$. Periodic timetables have been studied in the context of railway networks extensively [18, 22]. In the following we give a brief introduction of timetables that form the basis of our flight networks. A *flight timetable* is a tuple $\mathbb{T} := (\mathcal{C}, \mathcal{A}, \mathcal{F}, \zeta, \Pi)$ where $\mathcal{C}$ is a set of *elementary connections*, $\mathcal{A}$ a set of *airports*, $\mathcal{F}$ a set of *flights* and Π the *time period*. Additionally, $\zeta : \mathcal{A} \to \mathbb{Z}$ is a function which maps each airport to the *timezone* it belongs to. In our data, timezones are represented as UTC (Universal Time, Coordinated) offset from `UTC+0` with the same resolution as time points in general. An elementary connection $c \in \mathcal{C}$ is a tuple $c = (F, A_1, A_2, \tau_1, \tau_2)$ which is interpreted as flight $F \in \mathcal{F}$ departing at airport $A_1 \in \mathcal{A}$ at time τ_1 and arriving at airport $A_2 \in \mathcal{A}$ at time τ_2. Note that τ_1 and τ_2 are time points relative to the timezone of the airports A_1 and A_2. The *length* $\text{len}(c)$ of an elementary connection $c \in \mathcal{C}$ is then derived by stripping off the timezone offset $\tau_i' := \tau_i - \zeta(A_i) \mod \Pi$ for both $i = 1, 2$ and computing the length between the time points τ_1' and τ_2' with respect to the time period Π.

3 Modeling Issues

In basic, flight timetables are very similar to *railway timetables* as introduced in [18, 22]. In order to obtain a graph, two approaches exist for railway timetable information. The *time-expanded approach* rolls out the time-dependencies of the timetable and yields a time-independent graph where each node represents an event of the timetable and edges connect consecutive events. Their constant edge weight is depicted as the time duration (e.g., the length of one specific elementary connection) of its respective events. On the other hand, the *time-dependent approach* carries the time-dependencies of the timetable over to the graph. This results in time-dependent connection-edges where edge weights correspond to travel time functions of several trains sharing the same edge. While the former approach allows for more flexible modeling, the latter yields much smaller graph sizes which is important in the context of multi-modal route planning where overall graphs can become huge. For that reason, in this work we focus on engineering the time-dependent approach for modeling flight timetables.

3.1 Applying Railway Models

Several time-dependent railway models exist for efficient and realistic railway timetable information. The *condensed model* as introduced in [6] represents the adjacencies of the underlying network. Since this model does not account for transfer costs at stations, it has been extended to the realistic time-dependent model in [21].

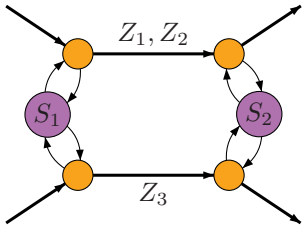


Fig. 1: Illustration of the time-dependent railway model when assuming a constant transfer time for each station with two stations served by two routes (with trains Z_1, Z_2 and Z_3, respectively).

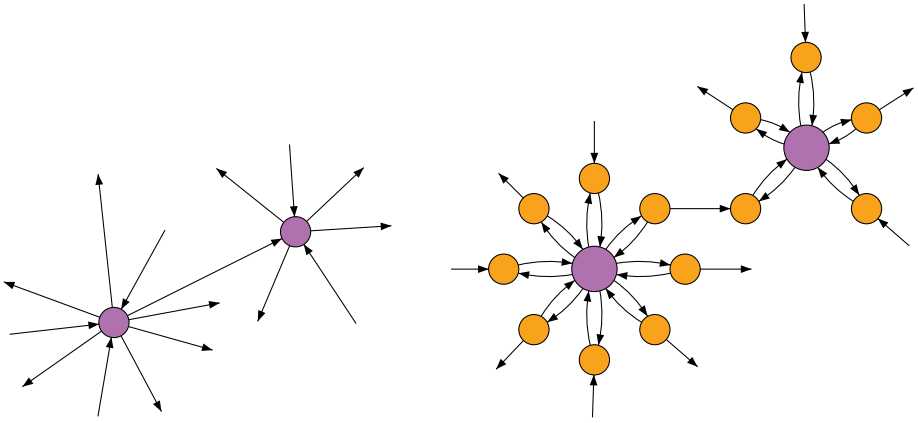
Briefly summarized, in a first step, the set of trains (in our case the set of flights $\mathcal{F}$) is divided into a set of *routes* $\mathcal{R}$. By these means, two trains (flights $F_1, F_2 \in \mathcal{F}$) are considered equivalent if they both share the exact same sequence of stations (airports $[A_1, \ldots, A_k]$). The graph is constructed by introducing a *station node* for every station (airport) and a *route node* for every route that runs through the specific station (airport). Edges from station to route nodes depict the constant transfer time while edges from route nodes to station nodes are modeled with zero cost. Connection edges are inserted between route nodes of the same route and are weighted by time-dependent travel-time functions depicting the travel time of trains running along the specific route. See Figure 1 for a small example.

The model can be extended further to account for variable transfer times between trains of different routes. This is achieved by introducing edges between each pair of route nodes r_1, r_2 at one station weighted by the time required to change from a train of route r_1 to a train of route r_2.

Drawbacks. Using the realistic time-dependent railway model on flight timetables yields several drawbacks which eventually lead to both inaccurate modeling regarding realism as well as unnecessarily large graphs, and thus, higher query times.

Routes. In flight timetables all routes have length 1, since almost all flights have no intermediate stops. In the rare case of flights serving a sequence $S = [A_1, \ldots, A_k]$ of airports, our flight timetables account for direct flights for each pair (A_i, A_j) with $i < j$ of airports (each possible subsequence of airports is modeled by a direct flight). As a conclusion, all routes are of length 1.

Regarding the number of nodes per airport in the graph, for each airport $A \in \mathcal{A}$ there is one route node per airport where at least one flight reaches to and also one route node per airport where at least one flight arrives from. Basically, the number of route nodes per airport can be bounded by 2 times the number of neighbors of A. This immediately leads to another drawback.



(a) Network structure (b) Resulting graph using the railway model

Fig. 2: Illustrating the high number of nodes and edges generated by the time-dependent railway model using a small example of two airports. Since all routes have length 1, for each neighbor in the network structure, a dedicated route node is inserted in the graph.

High Number of Neighbors. Whereas in railway networks the number of neighbors in the station graph for each station is relatively small (less than 5 for most of the stations [9]), airports tend to have lots more neighbors (cf. Section 5) due to the many direct flights. Combining this observation with the previous issue, we end up having unnecessarily many route nodes per airport. See also Figure 2 for an illustration of the high node and edge count when using route nodes for each flight at an airport.

Procedures at Airports. Most importantly, procedures at airports differ from procedures in train stations making the realistic railway model somewhat unrealistic. For example, boarding a flight at the departure airport including check-in involves more time than switching flights which may only require us to walk from one gate to another. Thus, at least two different types of times per airport are desirable: *Check-in time* and *transfer time*.

Another issue that should be reflected by the model is a third type of time for getting off at the destination airport. This *Check-out time* should cope for customs and baggage claim and is usually smaller than the *Check-in time*. While in principle the railway model could account for that by adjusting the edge weights of edges connecting route nodes to station nodes, incorporating a dedicated transfer time can only be achieved by inserting 'transfer edges' between all route nodes, yielding $\Theta(\mathcal{N}(A)^2)$ many edges where $\mathcal{N}(A)$ depicts the number of neighbors of an airport $A \in \mathcal{A}$. Because of the high number of neighbors this approach is infeasible. These problems lead us to proposing a family of new models for flight timetables with incrementing flexibility.

3.2 Tailored Models for Flight Timetables

The basis of our flight models is a flight timetable $\mathbb{T} = (\mathcal{C}, \mathcal{A}, \mathcal{F}, \zeta, \Pi)$. Furthermore, we introduce three different time functions to model the various procedures in an airport as depicted above.

- Check-in time $\mathcal{T}^{\text{ci}} : \mathcal{A} \to \mathbb{R}_0^+$.
  This accounts for the whole process from arriving at the airport until the departure of the plane composed of checking-in, passing security checks and also the accounted waiting time at the gate plus the boarding time of the plane.
- Check-out time $\mathcal{T}^{\text{co}} : \mathcal{A} \to \mathbb{R}_0^+$.
  This accounts for the reverse process: Leaving the plane, passing customs while leaving the gate area and finally the time required to claim baggage.
- Transfer time $\mathcal{T}^{\text{tr}} : \mathcal{A} \to \mathbb{R}_0^+$.
  This time accounts for the time transferring between two planes. Usually, this only involves leaving the plane, walking to another gate and boarding the new plane.

Note that we assume that all three time functions do not depend on the specific flights. In favor of more flexibility, this assumption is weakened in the second and third versions of our model.

Level I: Constant-Time Model. The Level I Model uses the time functions exactly as defined above. For each airport $A \in \mathcal{A}$ we insert a super node into the graph called *terminal node*. Since all flights either begin or end at the airport, we insert two more nodes per airport: A *departure node* which resembles flight departures, and an *arrival node* to model arrivals.

Edges are created in the following way. There are three edges within each airport. A *check-in edge* is inserted from the terminal node to the departure node and its weight is set to $\mathcal{T}^{\text{ci}}(A)$. A *check-out edge* from the arrival node to the terminal node with weight $\mathcal{T}^{\text{co}}(A)$ is inserted and finally a *transfer edge* from the arrival node to the departure node with weight $\mathcal{T}^{\text{tr}}(A)$ is created.

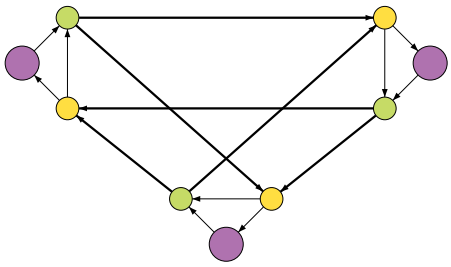


Fig. 3: Level I Model. Terminal nodes are purple, departure nodes green and arrival nodes yellow. Bold edges are time-dependent and model flights between the airports while the thin time-independent edges allow for check-in, check-out and transfers within the airports.

The actual flights are modeled as *flight edges* from the departure node of airport A_1 to the arrival node of airport A_2 if and only if there is at least one

elementary connection from A_1 to A_2 in the timetable. The edge weight is time-dependent and interpolation points are created for each elementary connection $c = (F, A_1, A_2, \tau_1, \tau_2)$ with departure time τ_1' and travel time $\text{len}(c)$.

An example of the Level I Model is shown in Figure 3. While this model yields very small graph sizes its drawback is the assumption that check-in, check-out, and transfer times are constant for all flights. This is addressed by the Level II Model.

Level II: Flight-Class Model. To account for more flexible check-in, check-out, and transfers within airports, we augment the definitions of $\mathcal{T}^{\text{ci}}$, $\mathcal{T}^{\text{co}}$ and $\mathcal{T}^{\text{tr}}$ to cope with different flight classes.

Similarly to the concept of routes in the realistic time-dependent railway model, we partition the set of flights $\mathcal{F}$ into different *flight classes*. The set of flight classes is denoted by $\mathfrak{C}$. The equivalence relation $\sim$ on the set of flights according to which two flights are put into the same class is arbitrary. An example might be $F_1 \sim F_2 \Leftrightarrow F_1$ and F_2 are operated by the same airline alliance.

With flight classes defined, the time functions are extended as follows. The check-in and check-out time functions are extended to $\mathcal{T}^{\text{ci}} : \mathcal{A} \times \mathfrak{C} \to \mathbb{R}_0^+$, and $\mathcal{T}^{\text{co}} : \mathcal{A} \times \mathfrak{C} \to \mathbb{R}_0^+$. The transfer-time function is extended to operate on pairs of classes $\mathcal{T}^{\text{tr}} : \mathcal{A} \times \mathfrak{C} \times \mathfrak{C} \to \mathbb{R}_0^+$ to account for transfers between flights of arbitrary pairs of flight-classes.

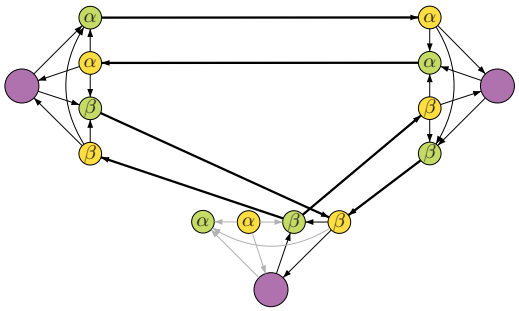


Fig. 4: The Level II Model with 3 airports and 2 classes α and β. The bottom airport has no incident flights of class α, thus, the respective nodes and (gray) edges can be omitted

The Level I Model is modified as follows. Let $A \in \mathcal{A}$ denote an airport. We insert $k := |\mathfrak{C}|$ departure resp. arrival nodes—one for each flight class $\mathfrak{c}_i \in \mathfrak{C}$. The departure and arrival nodes are connected to the terminal node by check-in and check-out edges like in the Level I Model. As edge weights we use $\mathcal{T}^{\text{ci}}(A, \mathfrak{c}_i)$ and $\mathcal{T}^{\text{co}}(A, \mathfrak{c}_i)$ for each of the classes. To incorporate transfers, for each pair $\mathfrak{c}_i, \mathfrak{c}_j$ of flight-classes we insert a transfer edge from the arrival node of class $\mathfrak{c}_i$ to the departure node of class $\mathfrak{c}_j$ weighted with $\mathcal{T}^{\text{tr}}(A, \mathfrak{c}_i, \mathfrak{c}_j)$. By this, we generate $O(k^2)$ edges. Finally, the time-dependent flight edges between two airports A_1 and A_2 are inserted with respect to the correct classes, i.e., if the flight is of class $\mathfrak{c}$, the departure node belonging to $\mathfrak{c}$ at A_1 is used as tail while the arrival node of the same class at A_2 is used as head of the edge. Interpolation points on the functions of the flight edges are created the same way as in the Level I Model.

In order to avoid the creation of unnecessary nodes, at each airport A we can omit the creation of departure and arrival nodes (and their incident edges) which belong to flight classes that do not contain any outgoing resp. incoming connections from/to the airport A. Figure 4 shows a small example consisting of two flight classes α and β.

Level III: Variable-Time Model. This is the most flexible model, however, some of the drawbacks worked out for railway models recur. The Level II Model is generalized further by assuming that each flight $F \in \mathcal{F}$ belongs to a distinct flight class. Thus, the set $\mathfrak{C}$ of flight classes consists of singleton sets and it holds that $|\mathfrak{C}| = |\mathcal{F}|$. By these means, we are able to model individual check-in, check-out, and transfer times for each (pair of) flight(s).

On the downside, the size of the graph becomes very large. For an airport $A \in \mathcal{A}$ let $\mathcal{C}(A)$ denote the set of elementary connections either departing or arriving at A. Then this model yields $\Theta(|\mathcal{C}(A)|)$ nodes and $\Theta(|\mathcal{C}(A)|^2)$ edges per airport. Since in general it holds that $|\mathcal{C}(A)| > |\mathcal{N}(A)|$, graphs generated by this model turn out even larger than using the realistic time-dependent railway model.

Level I and III Models as a Special-Case. We like to point out, that both the Level I and Level III Models can be seen as special cases of the Level II Model. In the case of $|\mathfrak{C}| = 1$, i.e., we only have one flight class, we obtain the Level I Model, while in the case of $|\mathfrak{C}| = |\mathcal{F}|$ we obtain the Level III Model as described above. Thus, by adjusting the number of flight classes we are able to control the flexibility of the resulting model in a continuous way. However, for real world scenarios a very limited number of flight classes seems sufficient (for example, using each major flight alliance as a dedicated class, since transfers within flights of the same airline alliance can be usually processed faster).

4 Route Planning in Flight Networks

In this section, we show how to compute best connections with the models introduced in Section 3.

4.1 Quickest Connections (Earliest Arrival Problem)

In the Earliest Arrival Problem, given source and destination airports S and T as well as a departure time $\tau_S < \Pi$, we ask for an itinerary from S to T arriving at T as early as possible and departing at S no earlier than τ_S. The straightforward approach to compute the earliest arrival for a given departure time is to run plain Dijkstra on any of the above proposed model. We simply insert the terminal node of the desired departure airport S into a priority queue and run Dijkstra's algorithm until we settle the terminal node of the requested arrival airport T. However, especially in flight networks, we are often interested for all 'optimal' connections during a whole day (resp. time period). This can

be done by a so-called profile query [8]. Such queries determine the travel time function between two airports for the full time period Π. This can be achieved by a label-correcting variant of DIJKSTRA's algorithm. The main difference to plain DIJKSTRA is that we propagate functions instead of constants through the network (cf. [7, 8] for details). Note that by this procedure, the algorithm loses its label-setting property, i.e., a node may be settled more than once during one run of the algorithm. The departure times of the optimal connections are then exactly the local minimums of the computed travel time function between S and T.

4.2 Multi-Criteria Connections

Up to now, we only showed how to compute quickest connections in flight networks. However, we might be willing to accept slightly longer routes if the costs are less or the number of transfers is smaller. A common approach to obtain such better routes is to compute Pareto routes. In this work, we run multi-criteria profile searches, i.e., we obtain Pareto connections between two stations for the full time period. Besides travel time, we use the number of transfers and costs as additional optimization criteria.

The Pareto connections between two airports can be obtained by a generalized version of DIJKSTRA's algorithm, similar to as introduced in [13, 14]. At each node u, we maintain a list of labels $\texttt{list}(u)$. In our case, a label contains a travel time function, the number of transfers, and the costs of the tentative journey. The list at the source node s is initialized with a label $L_s := (0, \ldots, 0)$. We insert L_s into the priority queue. Then, in each iteration, we extract the label with the smallest lower bound of its respective travel time function. Let u be the associated node of the label. Then for all outgoing edges $(u, v) \in E$ a temporary label L_v is generated depicting the journey to v via u. If L_v is not dominated by any of the labels in $\texttt{list}(v)$, we add L_v to $\texttt{list}(v)$, add L_v to the priority queue, and remove all labels from $\texttt{list}(v)$ that are dominated by L_v. We may stop the query as soon as the priority runs empty or all labels in the priority queue are dominated by all labels in $\texttt{list}(t)$.

Rules of Dominance. In order to be able to run the algorithm described above, we require to compare labels. We say that one label (consisting of several components) *dominates* another label if it is better with respect to at least one component and not worse respect to the remaining components. Note that in our case, one component of our labels is a function. A travel time function f is better than a function g if $f(x) < g(x)$ holds for all $x < \Pi$. For more details on dominance, we refer the interested reader to [12].

Generating Costs. Unfortunately, real-world pricing information was not available to us. Moreover, using arbitrary flight-costs per flight in time-dependent graphs may result in non-FIFO networks making the computation of shortest paths $\mathcal{NP}$-hard [20]. Thus, we restrict ourselves to generated constant costs per

edge. We generate pricing information as follows. For each flight edge $(u, v) \in E$ we compute price $E \to \mathbb{R}^+$ according to

$$\text{price}(e) := \text{fee}(u) + \text{fee}(v) + \text{fuel}(e) + \text{charge}(e), \tag{1}$$

where $\text{fee}(\cdot)$ depicts an airport fee, $\text{fuel}(e)$ costs for fuel along the edge e and $\text{charge}(e)$ the amount of money charged by the flight operator. The airport fee is computed by

$$\text{fee}(A) := \left(\alpha_f + \beta_f |\mathcal{F}(A)|\right) \cdot \rho(A), \tag{2}$$

where α_A is a general base fee, $\mathcal{F}(A)$ the number of flights departing/arriving at A, and β_A a constant coefficient. Furthermore, we perturb the costs by 25% by choosing $\rho(A) \in [0.75, 1.25]$ uniformly at random for each airport. Fuel costs are computed by

$$\text{fuel}(e) := \gamma \cdot \sqrt{\text{dist}_{\text{geo}}(e)}, \tag{3}$$

where γ is a coefficient and $\text{dist}_{\text{geo}}(e)$ is the geodesic length of the flight edge (we use the GRS80-ellipsoid [16] with geographic coordinates for computing distances). Finally, $\text{charge}(e)$ is computed by

$$\text{charge}(e) := \sqrt{\alpha_c + \beta_c \, \text{dist}_{\text{geo}}(e)} \cdot \rho(e). \tag{4}$$

Again, α_c is a base charge, β_c a constant coefficient. However, to model more varying charges we perturb the costs by 50% by choosing $\rho(e) \in [0.5, 1.5]$ uniformely at random.

Our final prices are generated by instantiating $\alpha_f := 15$, $\beta_f := 0.1$, $\gamma := 0.2$, $\alpha_c := 30$, $\beta_c := 0.5$ resulting in flight costs between €60 for very short and up to €1500 for long distance (intercontinental) flights.

4.3 Storing Distance Tables

During our experimental studies, it turned out that the resulting graphs deriving from our level II Model are so small, that it is feasible to do a all-pair-shortest-path preprocessing. This even holds for multi-criteria route planning. In the following, we shortly explain how to preprocess the distance table in a multi-criteria scenario, in case of single-criteria, we proceed analogously.

The preprocessing can be done in a straightforward manner. We maintain a distance table with size $|\mathcal{A}| \times |\mathcal{A}|$. For each airport A_i, we run a full Pareto-Dijkstra as described above. This results in a set of labels for each airport A_j depicting the Pareto-connections from A_i to A_j, which we store at the corresponding place in the distance table. After having performed this step for any airport, the distance table contains the Pareto connections for any pair of airports. Hence, running a query is then reduced to a table-lookup in the distance table.

5 Experiments

We conducted our experiments on one core of an AMD Opteron 2218 running SUSE Linux 10.3. The machine is clocked at 2.6 GHz, has 32 GB of RAM and 2 x 1 MB of L2 cache. The program was compiled with GCC 4.2, using optimization level 3. Our implementation is written in C++ using solely the STL and Boost at some points. As priority queue we use a binary heap.

Inputs. Our inputs derive from (publicly available) timetables of two major flight alliances, which we crawled from the companies webpages. The first is of StarAlliance [24] from November 2008 containing 20 888 flights between 965 airports. The latter is of Oneworld [19] (also November 2008) and contains 8 602 flights between 621 different airports. To make use of the Level II Model, we also use a combined timetable which contains flights of both, StarAlliance and Oneworld. The resulting timetable contains 29 490 flights and 1 172 airports.

Table 1 reports figures of the parameters of our input data. Besides the number of airports and flights we show the average degree on the condensed network (nodes equal airports and an edge (u, v) is inserted, iff. there is at least one flight going from u to v). For comparison, we also provide figures for a typical railway timetable (Ger-Rail) consisting of all trains in Germany operated by the Deutsche Bahn in the winter period 2000/2001. We observe that the average degree is significantly higher in flight timetables, while the maximum degree is even up to 5 times larger. Moreover, in Figure 5 we show a straight line visualization of our combined timetable. Blue spots depict airports (light spots are not served by the timetable and are only drawn for orientation). The size of the nodes reflects the number of flights departing and arriving at the specific airports.

Methodology. In the following, we report query performance on each of our models regarding both profile search and multi-criteria search using travel-time, number of transfers and pricing as criteria (cf. Section 4). We evaluate the query performance by running 1 000 random queries, picking source and destination airports uniformly at random. We report the number of settled nodes, relaxed edges and the average time per query.

Table 1: Figures for our input data. We use timetables of StarAlliance and Oneworld as well as a combined timetable of both alliances. As comparison, we also provide data for a railway timetable consisting of all German trains operated by Deutsche Bahn.

Timetable	# Airports	# Flights	Avg. Deg.	Max. Deg.
StarAlliance	965	20 888	13.35	175 (FRA)
Oneworld	621	8 602	8.86	152 (DWF)
Combined	1 172	29 490	14.52	192 (ORD)
	# Stations	# Conns		
Ger-Rail	6 822	554 996	5.41	37 (Leipzig Hbf)

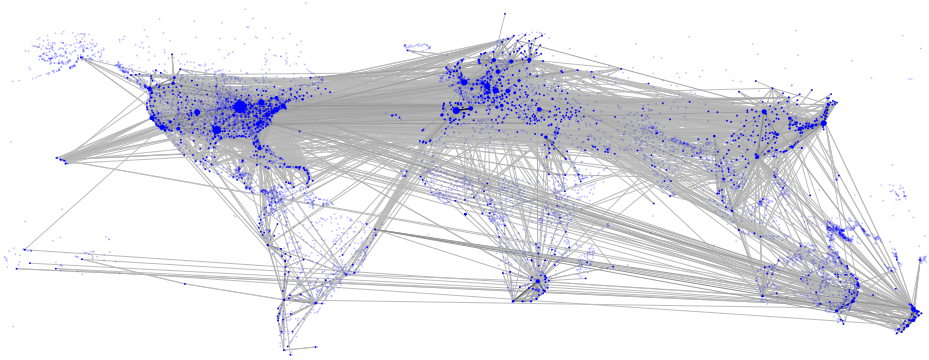


Fig. 5: Flight network composed of timetables from StarAlliance and Oneworld.

Regarding our table-lookup algorithm, we report preprocessing effort as the amount of additional required space in Megabytes as well as preprocessing time. Since table-lookups do not involve settled nodes and relaxed edges, we restrict ourselves to query time together with the speed-up compared to the default algorithm. Moreover, we increase the number of random queries to 10 000 0000 and report the query time by measuring the whole execution time of all queries divided by the number of queries.

5.1 Size of the Models

Table 2 reports figures on the graph parameters of the different models introduced in Section 3. For each of our inputs we apply the Level I, Level II and Level III flight models, whereas regarding the Level II Model we use each flight alliance as a separate flight class. Moreover, for comparison, we also apply the time-dependent railway model with constant transfer times [22]. Besides reporting the total number of nodes and edges of the resulting graphs, we also present the average number of flights per edge (only taking flight edges into account).

Applying the railway model to our flight timetables yields graphs of 13 849 nodes with 32 210 edges regarding the StarAlliance timetable, 6 123 nodes with 13 755 edges regarding the Oneworld timetable, and 18 184 nodes with 42 530

Table 2: Comparison of the sizes in number of nodes, edges and flights per edge. The latter only refers to flight edges (not intra-airport edges)

	StarAlliance			Oneworld			Combined		
Model	Nodes	Edges	Fl/Edge	Nodes	Edges	Fl/Edge	nodes	Edges	Fl/Edge
Level I	2 719	8 986	2.52	1 834	4 557	2.25	3 397	11 785	2.68
Level II	2 719	8 986	2.52	1 834	4 557	2.25	4 139	14 286	2.36
Railway	13 849	32 210	1.43	6 123	13 755	1.41	18 184	42 530	1.46
Level III	42 741	3 085 752	1.00	17 825	1 234 362	1.00	60 152	6 072 836	1.00

edges on the combined timetable. On all three instances graph sizes decrease significantly when we use the Level I and II Models: in each timetable the number of nodes and edges is between 3 and 5.4 times lower while incorporating more realistic airport procedures (cf. Section 3). Note that the Level I and II Model graphs are of equal size on the StarAlliance and Oneworld instances since they only contain one flight class. However, on the combined instance switching from the Level I to the more flexible Level II Model yields only a small increase regarding graph size (4 139 compared to 3 397 nodes and 14 286 edges compared to 11 785 edges).

Concerning the Level III Model, graph sizes increase dramatically. While the increase in number of nodes compared to the Level II Model is between 10 and 15 times, the number of edges increases up to 6 072 836 on our combined timetable. This is due to the fact that for each elementary connection on each airport one dedicated node is created, and these departure respective arrival nodes become fully interconnected yielding a quadratic number of edges in the number of incident flights at each airport. The fact that for each flight a separate (time-dependent) flight-edge is created, is also reflected by the number of flights per edge, which is exactly 1 in the Level III Model.

5.2 Query Performance

Label Correcting Algorithms. Regarding profile and multi-criteria queries, we use a label correcting algorithm (cf. Section 4) which may settle nodes multiple times during one run. Moreover, we use travel-time, number of transfers and pricing information as optimization criteria. In Table 3 we report the number of settled nodes, relaxed edges and the average time per query on each of our models.

As expected, figures roughly concur with the graph sizes from Table 2. Using the railway model yields query times of 264.86 ms settling 71 673 nodes. On the Level I Model we are able to reduce the query time to 47.5 ms while only settling 8 426 nodes. Applying the Level II Model only yields a mild decrease in performances to 68.68 ms settling 11 110 nodes which is still almost 4 times faster than the time-dependent railway model.

Table 3: Query performance of our models using label correcting algorithms for both profile- and multi-criteria searches. Query performance is evaluated by running 1 000 queries with source and destination airports picked uniformely at random.

	Profile			Multi-Criteria		
Model	Settled Nodes	Relaxed Edges	Time [ms]	Settled Nodes	Relaxed Edges	Time [ms]
Level I	8 426	41 462	47.55	23 825	104 213	215.74
Level II	11 110	53 477	68.68	31 491	137 068	305.31
Railway	71 673	171 924	264.86	184 516	435 062	1 126.50
Level III	133 083	5 739 353	4 805.60	673 295	32 180 968	109 666.66

Regarding multi-criteria search, we are able to enumerate all Pareto optimal solutions in under a second's time on both the Level I and Level II Models (215 ms and 305 ms, respectively). However, both algorithms perform significantly worse on the much larger Level III Model resulting in query times of 4.8 seconds for profile queries and almost 2 minutes for multi-criteria queries.

Table-Lookups. The very small graph sizes of our flight networks allow precomputation of full distance tables between all airports. Regarding profile search, we store travel-time functions for each pair of airports, while we store all Pareto solutions when using multi-criteria search.

Table 4: Accelerating queries by table-lookups. We report the additional space required as well as preprocessing time. On the query side we report the query time as well as the speed-up compared to our label correcting algorithm from Table 3.

	Profile Table-Lookup				Multi-Crit. Table-Lookup			
Model	Space [MiB]	Prepro [m:s]	Time [μs]	Speed-Up	Space [MiB]	Prepro [m:s]	Time [μs]	Speed-Up
Level I	45.65	0:58	0.41	115 973	282.91	4:35	2.85	75 697
Level II	45.65	1:21	0.40	171 710	297.01	6:14	2.97	102 799
Railway	45.65	5:01	0.37	715 841	288.58	21:37	2.83	398 056
Level III	45.65	60:28	0.41	11 720 969	433.28	2618:23	4.37	25 095 345

Profile Search. Table 4 reports both preprocessing effort and query performance on the combined timetable network for each of our models. For profile queries the additional space required for each model is 45.65 MiB (note that we compute distances between pairs of airports, thus, the required space is independent of the number of nodes). Compared to the small size of our networks, 45.65 MiB may seem fairly much. However, from the perspective of multi-modal route planning, this additional effort is almost negligible, since the space consumption of all data is dominated by the significantly larger road network and also by additional data required for multi-modal speed-up techniques [10].

Regarding the preprocessing time, we are able to compute the full distance table of travel-time functions between 1 minute on the Level I Model and 1 hour on the Level III Model. As a result, we are able to execute random profile queries in approximately 0.4 μs time yielding a speed-up of over 11 Million on the Level III Model. Note, that the query times are independent of the graph size, since the graph is not used in the query algorithm.

Multi-Criteria Search. For multi-criteria search the required space for storing the distance table increases with the complexity of the model and requires from 282.91 MiB (Level I Model) to 433.28 MiB (Level III Model) space. In contrast to profile-search distance tables, here the number of entries in the table for each pair

of airports depends on the model for the following reason. Since we do not store flight costs per actual flight, but a combined price per flight edge (cf. Section 4), having less flights per edge (cf. Table 2) allows us to assign a greater variety of different costs for flights between the same two airports. As a consequence, the number of Pareto optimal solutions increases, hence, requiring more space. The extreme case is the Level III Model, where each flight has its own designated flight edge, and thus, allows the most realistic cost assignments (we are actually able to assign a different price for each flight). Preprocessing time for distance tables increases with the complexity of the models and is between 4.5 minutes on the Level I Model and almost two days on the Level III Model. Again, we like to point out the insignificant deterioration in preprocessing performance of the Level II Model compared to the Level I Model. Query times are in the scale of a few microseconds on all models: Enumerating all Pareto solutions for random queries requires 2.85 μs on the Level I Model, 2.97 μs on the Level II Model and 4.37 μs on the Level III Model. Again, the increase in query time is explained by the bigger number of Pareto solutions with increasing model complexity.

6 Conclusion

In this work, we introduced how to model flight networks as graphs such that we are able to compute best connections efficiently. By showing that known models for railways yield a significant performance penalty, we justify our new model. Moreover, we showed how to generate flight costs if data is missing. It turns out that our model yields such small graphs making it feasible to compute full Pareto distance tables making multi-criteria route planning in flight networks a matter of microseconds. More precisely, we are able to perform point-to-point profile and multi-criteria queries within a few microseconds with space requirements of 43.65 MiB (profile-search) and up to 433.28 MiB (multi-criteria search). While the Level III Model is the most flexible, it turns out that in the case of using flight alliances as flight classes, the Level II Model is sufficiently realistic while yielding significantly smaller graphs, and thus, faster query times. However, we like to point out, that in the case of our table-lookup algorithm the graph is no longer required as input. Hence, it becomes feasible to apply a two step approach for flight timetable information: Use a high detailed model (i.e., the Level III Model) for modeling the timetable in the most flexible way, and obtain the distance table in a second step. Queries can then be answered solely using the distance table, thus, no longer requiring the large flight graphs as input data.

Regarding future work, it would be interesting to integrate traffic days into our model. Moreover, we would like to add low-cost carriers to our data. However, such companies tend to serve only very small airports which are far away from the main hubs. Hence, we expect the network to be disconnected such that multi-modal route planning becomes even more important in such a scenario. On the technical side, we are optimistic that with the insights gained in this work, we may extend our recent work on multi-modal route planning [10] to a full multi-modal variant of Transit-Node Routing [3, 23, 4, 5].

References

1. C. Barrett, K. Bisset, R. Jacob, G. Konjevod, and M. V. Marathe. Classical and Contemporary Shortest Path Problems in Road Networks: Implementation and Experimental Analysis of the TRANSIMS Router. In R. H. Möhring and R. Raman, editors, *Proceedings of the 10th Annual European Symposium on Algorithms (ESA'02)*, volume 2461 of *Lecture Notes in Computer Science*, pages 126–138. Springer, 2002.
2. C. Barrett, R. Jacob, and M. V. Marathe. Formal-Language-Constrained Path Problems. *SIAM Journal on Computing*, 30(3):809–837, 2000.
3. H. Bast, S. Funke, and D. Matijevic. TRANSIT Ultrafast Shortest-Path Queries with Linear-Time Preprocessing. In C. Demetrescu, A. V. Goldberg, and D. S. Johnson, editors, *Shortest Paths: Ninth DIMACS Implementation Challenge*, DIMACS Book. American Mathematical Society, 2009. Accepted for publication, to appear.
4. H. Bast, S. Funke, D. Matijevic, P. Sanders, and D. Schultes. In Transit to Constant Shortest-Path Queries in Road Networks. In *Proceedings of the 9th Workshop on Algorithm Engineering and Experiments (ALENEX'07)*, pages 46–59. SIAM, 2007.
5. H. Bast, S. Funke, P. Sanders, and D. Schultes. Fast Routing in Road Networks with Transit Nodes. *Science*, 316(5824):566, 2007.
6. G. Brodal and R. Jacob. Time-dependent Networks as Models to Achieve Fast Exact Time-table Queries. In *Proceedings of ATMOS Workshop 2003*, pages 3–15, 2004.
7. B. C. Dean. Continuous-Time Dynamic Shortest Path Algorithms. Master's thesis, Massachusetts Institute of Technology, 1999.
8. D. Delling. Time-Dependent SHARC-Routing. In *Proceedings of the 16th Annual European Symposium on Algorithms (ESA'08)*, volume 5193 of *Lecture Notes in Computer Science*, pages 332–343. Springer, September 2008. Best Student Paper Award - ESA Track B.
9. D. Delling, T. Pajor, and D. Wagner. Engineering Time-Expanded Graphs for Faster Timetable Information. In *Proceedings of the 8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'08)*, Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, September 2008.
10. D. Delling, T. Pajor, and D. Wagner. Accelerating Multi-Modal Route Planning by Access-Nodes. In A. Fiat and P. Sanders, editors, *Proceedings of the 17th Annual European Symposium on Algorithms (ESA'09)*, Lecture Notes in Computer Science. Springer, September 2009. Accepted for publication, to appear.
11. E. W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1:269–271, 1959.
12. Y. Disser, M. Müller–Hannemann, and M. Schnee. Multi-Criteria Shortest Paths in Time-Dependent Train Networks. In C. C. McGeoch, editor, *Proceedings of the 7th Workshop on Experimental Algorithms (WEA'08)*, volume 5038 of *Lecture Notes in Computer Science*, pages 347–361. Springer, June 2008.
13. P. Hansen. Bricriteria Path Problems. In G. Fandel and T. Gal, editors, *Multiple Criteria Decision Making – Theory and Application –*, pages 109–127. Springer, 1979.
14. E. Q. Martins. On a Multicriteria Shortest Path Problem. *European Journal of Operational Research*, 26(3):236–245, 1984.

15. A. O. Mendelzon and P. T. Wood. Finding Regular Simple Paths in Graph Databases. *SIAM Journal on Computing*, 24(6):1235–1258, 1995.
16. H. Moritz. Geodetic Reference System 1980. *Journal of Geodesy*, 66(2):187–192, June 1992.
17. M. Müller–Hannemann and M. Schnee. Finding All Attractive Train Connections by Multi-Criteria Pareto Search. In *Algorithmic Methods for Railway Optimization*, volume 4359 of *Lecture Notes in Computer Science*, pages 246–263. Springer, 2007.
18. M. Müller–Hannemann, F. Schulz, D. Wagner, and C. Zaroliagis. Timetable Information: Models and Algorithms. In *Algorithmic Methods for Railway Optimization*, volume 4359 of *Lecture Notes in Computer Science*, pages 67–90. Springer, 2007.
19. Oneworld Management Ltd. http://www.oneworld.com, 1999.
20. 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.
21. E. Pyrga, F. Schulz, D. Wagner, and C. Zaroliagis. Towards Realistic Modeling of Time-Table Information through the Time-Dependent Approach. In *Proceedings of ATMOS Workshop 2003*, pages 85–103, 2004.
22. E. Pyrga, F. Schulz, D. Wagner, and C. Zaroliagis. Efficient Models for Timetable Information in Public Transportation Systems. *ACM Journal of Experimental Algorithmics*, 12:Article 2.4, 2007.
23. P. Sanders and D. Schultes. Robust, Almost Constant Time Shortest-Path Queries in Road Networks. In C. Demetrescu, A. V. Goldberg, and D. S. Johnson, editors, *Shortest Paths: Ninth DIMACS Implementation Challenge*, DIMACS Book. American Mathematical Society, 2009. Accepted for publication, to appear.
24. Star Alliance. http://www.staralliance.com, 1997.