Multi-criteria Shortest Paths in Time-Dependent Train Networks

Yann Disser[1], Matthias Müller–Hannemann[2], and Mathias Schnee[1]

[1] Technische Universität Darmstadt, Department of Computer Science,
Hochschulstraße 10, 64289 Darmstadt, Germany
{disser,schnee}@algo.informatik.tu-darmstadt.de
[2] Martin-Luther-Universität Halle-Wittenberg, Department of Computer Science,
Von-Seckendorff-Platz 1, 06120 Halle, Germany
muellerh@informatik.uni-halle.de

Abstract. We study the problem of finding all Pareto-optimal solutions in a multi-criteria setting of the shortest path problem in time-dependent graphs. This has important applications in timetable information systems for train schedules. We present a new prototype to solve this problem in a fully realistic scenario based on a multi-criteria generalization of Dijkstra's algorithm. As optimization criteria we use travel time and number of train changes, as well as a new criterion "reliability of transfers".

The performance of the prototype and various speed-up techniques are analyzed experimentally on a large set of real test instances. In comparison with a base-line implementation, our prototype achieves significant speed-up factors of 20 with respect to the number of label creations and of 138 with respect to label insertions into the priority queue. We also compare our prototype with a time-expanded graph model.

Keywords: shortest paths, time-dependent graphs, multi-criteria optimization, speed-up techniques, case study.

1 Introduction

In peak times the timetable information system of the German railway company Deutsche Bahn AG calculates over 1,600,000 connections per hour [1]. This demonstrates the importance that such systems have gained. It is obvious that efficient algorithms have to be used in order to cope with that large a demand. To achieve this kind of efficiency, the system currently in use by Deutsche Bahn AG applies rather restrictive heuristics. Therefore, optimality of the gained results cannot be guaranteed. Moreover, commercial systems usually apply single-criteria algorithms optimizing travel time only.

In recent years, several research efforts have demonstrated that *exact* single-criteria shortest path queries in train networks can be performed very efficiently due to powerful speed-up techniques. Multi-criteria shortest path search is much more challenging. Given two paths p and q, we say that p *dominates* q if and only if there is at least one criterion for which p has a better value than q and

C.C. McGeoch (Ed.): WEA 2008, LNCS 5038, pp. 347–361, 2008.

there is no criterion for which p has a worse value than q. A path is called *Pareto-optimal* if it is not dominated by any other path. Here, the usual goal is to find *all* Pareto-optimal solutions. In theory, there can be exponentially many Pareto-optima in the worst case, although in practice only a few are observed in a realistic setting [2]. But in contrast to single-criteria search, one cannot abort the search after finding a first optimal solution. In fact, even after finding all Pareto-optima, search algorithms require a substantial amount of time to find a certificate that no further solutions exist.

Related work. Two main approaches have been proposed for modeling timetable information as a shortest path problem: the *time-expanded* [3,4,5], and the *time-dependent* approach [6,7,8,9,10,11,5]. These models and algorithms are described in detail in a recent survey [12].

Pyrga et al. [5] have presented an extensive computational study comparing the time-expanded and the time-dependent graph for the earliest arrival problem. They also consider a bicritera search for all Pareto-optima with respect to travel time and number of transfers. However, in the time-dependent version they heavily exploit the fact that the number of transfers in Pareto-optimal solutions is usually fairly small. More specifically, Pyrga et al. reduce the bicriteria search to a sequence of single-criteria problems with a bounded number of transfers. They start by obtaining the lexicographically smallest optimum for the combination of earliest arrival and number of transfers. If this optimum uses T transfers, another T searches are performed bounding the number of transfers by $T-1,\ldots,0$. By excluding dominated results from all the obtained ones, all Pareto-optima are computed in this particular bicriteria scenario. This trick is neither well-suited for more than two criteria nor for criteria which may attain a large range of values.

The second and third author have designed a fully realistic multi-criteria prototype MOTIS (multi-criteria timetable information system) which is capable of answering queries in about one second on standard PCs [13]. The MOTIS system is currently based on a time-expanded graph due to the fact that it is much easier to model all side constraints arising in practice in this framework. However, the major drawback of time-expanded graphs in comparison to time-dependent models is the higher space consumption, in particular if highly-periodically operating regional mass transit has to be included. In addition, the time-dependent graph model is easier to adapt in case of dynamic graph changes due to train delays. This motivates our investigation of the time-dependent graph model in this paper.

Only a few months ago Bauer et al. [14] have presented an experimental study on speed-up techniques for timetable information systems. They observed that many of the recently developed speed-up techniques are much slower on graphs derived from timetable information than on road networks. Moreover, many single-criteria speed-up techniques rely on a simultaneous bidirected search from source and target. Such techniques are not applicable in train search applications since we only know the target station but not the time at which an arrival can be expected. A recently developed technique is the unidirectional routing algorithm

SHARC [15]. A time-dependent version of SHARC yields only approximations, but works well on road networks.

Our Contribution and Overview. To the best of our knowledge, no complete, realistic system has been built for exact multi-criteria search of all Pareto-optimal solutions in the time-dependent graph model. In [5], Pyrga et al. treat constant transfer times and traffic days, but other aspects of real timetables like foot-paths and special transfer rules are not considered. In this paper we describe a first prototype for multi-criteria search of all Pareto-optima within a fully featured, real timetable. Its search results are guaranteed to be optimal. We provide an extensive computational study showing the impact of several speed-up techniques. Even though the number of possible speed-up techniques is restricted severely in order to guarantee the optimality of all search results, the performance of our prototype is already comparable to time-expanded systems, but consumes much less space.

Most previous research (in particular [5]) concentrates on the earliest arrival problem from a given point in time. But here we focus on a many-source shortest path version because in a pre-trip search for train connections a user usually wants to specify a *time interval* in which his journey should start. This implies that we have to perform a simultaneous search from multiple starting times. In a time-expanded graph model this can be handled very easily: One simply adds a "super-source" and edges of length zero to all start events, thereby reducing the search to a single-source search. In time-dependent graphs, however, solving the many-source shortest path problem is more subtle if travel time is used as an optimization criterion. Consider two subpaths from the source to some intermediate node. Then, path p_1 with start time s_1 and travel time t_1 dominates another path p_2 with start time s_2 and travel time t_2 with respect to travel time only if $t_1 < t_2$ *and* $s_1 \geq s_2$. Otherwise both paths are incomparable. This leads to weaker dominance during search than for the earliest arrival problem, and consequently to more non-dominated solutions which can be offered to customers. It is therefore remarkable that we still achieve quite a reasonable performance.

Our approach can easily be extended to further criteria. In order to exemplify this, the "reliability of transfers" is introduced as an additional criterion. The reliability of transfers is a property of a connection that captures the probability of catching all trains within it. Since possible train delays cannot be ignored, such a criterion is of practical importance.

The remainder of this paper is organized as follows. In Section 2, we introduce the time-dependent graph model and describe the adaptations needed in order to make it suited for fully realistic timetables. A modification of Dijkstra's algorithm that makes it capable of minimizing multiple criteria is introduced in Section 3. Several speed-up techniques that do not violate the optimality of the search results are proposed. The results of the experimental analysis of our time-dependent search system are presented in Section 4. We analyze the impact of the proposed speed-up techniques on performance. The final performance is then compared to a fully optimized search using a time-expanded graph. The

last aspect of our discussion covers the relationship between performance and the number of search criteria. Finally, Section 5 summarizes and gives an outlook on future work.

2 Realistic Time-Dependent Graph Model

In this section we will describe a time-dependent graph model as introduced in [5,10,11]. We will start off with a very basic time-dependent model and extend it in the following.

We assume the timetable to consist of a set $\mathcal{T}$ of trains, a set $\mathcal{S}$ of stations, and a set $\mathcal{E}$ of elementary connections. An *elementary connection* $e \in \mathcal{E}$ describes a connection between two adjacent train stations without intermediate stops. Such a connection contains a departure station $\text{from}(e) \in \mathcal{S}$, an arrival station $\text{to}(e) \in \mathcal{S}$, a departure time $\text{d}(e)$, and an arrival time $a(e)$. In addition to that, each elementary connection has several properties like train class, traffic days and train number. Each train $tr \in \mathcal{T}$ is an ordered list of elements of $\mathcal{E}$. A *train connection* is composed of an ordered list of elementary connections which must be consistent with the sequence of departure and arrival stations.

2.1 Basic Time-Dependent Model

For each station $S \in \mathcal{S}$ in the timetable there is a node $v(S) \in V$ in the basic time-dependent graph $G = (V, E)$. We call these nodes *station nodes*. There is an edge $e_{AB} = (v(A), v(B)) \in E$ if the set $\mathcal{E}_{AB} := \{e \in \mathcal{E} | \text{from}(e) = A \wedge \text{to}(e) = B\}$ is non-empty. The characteristics of all elementary connections in $\mathcal{E}_{AB}$ are attributed to this single edge e_{AB}. Each edge has multiple length functions, one for each optimization criterion. These length functions are time-dependent: depending on the time t at which the edge is to be used, different connections in $\mathcal{E}_{AB}$ may be favorable. In general, this is implemented with an iterator which computes edge lengths "on-the-fly" and returns all necessary variants with different characteristics.

If we only consider travel time and make the assumption that a connection $e_1 \in \mathcal{E}_{AB}$ *may not* overtake another connection $e_2 \in \mathcal{E}_{AB}$ in the sense that $\text{d}(e_1) \geq \text{d}(e_2)$ and $\text{a}(e_1) < \text{a}(e_2)$, then the connection with the earliest departure after time t is the one chosen from $\mathcal{E}_{AB}$. Its travel time length is precisely $\text{a}(\text{rel}(\mathcal{E}_{AB}, t)) - t$, where $\text{rel}(\mathcal{E}_{AB}, t) := \arg\min_{e \in \mathcal{E}_{AB}, \text{d}(e) \geq t} \text{d}(e)$ is the relevant connection in $\mathcal{E}_{AB}$ at time t.

2.2 Transfers

In the basic model, transfers between different trains are not modeled differently than two consecutive elementary connections with the same train. In order to allow for our search to count the number of transfers and in order to assign a duration to transfers, the model has to be extended as follows. We assume here for simplicity that a constant transfer time is provided for each station.

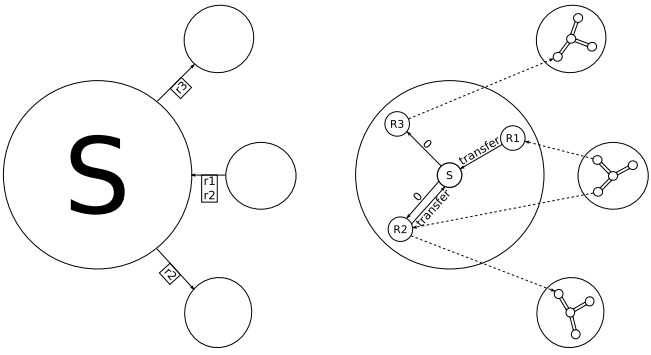

Fig. 1. Extension of a simple time-dependent graph (left) to support transfers. The timetable has three routes r1, r2, r3 so that the extended station (right) has three route nodes.

In order to still be able to take advantage of the fact that multiple elementary connections are modeled by a single edge, it is necessary to group train connections into *routes.* The set of routes forms a partition of $\mathcal{T}$ such that two connections are in the same route if and only if they share equal stations and properties. The departure and arrival times of two connections in the same route may differ as well as their traffic days. Using this partition, each station is represented by several *route nodes* in addition to its station node. The station node is used only to connect the route nodes and has no edges to nodes from other stations. The expanded model is depicted in Figure 1.

One route node is required for each route that arrives or departs at the station. For all connections in the same route, the corresponding route node plays the role of the station node in the basic model. The assumption that connections may not overtake each other can now be restricted to connections within a route. If we have overtaking elementary connections within a route, the route can simply be split up in order to separate the two elementary connections (and so we can get rid off this assumption). If the route has a connection that arrives at the station, an edge connecting the route node to the station node is introduced; if the route has a connection that departs from the station, an edge connecting the station node to the route node is introduced. One of these two edges needs to carry the transfer costs at the station and is called *transfer-edge*, the other has a transfer cost of 0. In the following we choose the edges from route nodes to station nodes as transfer-edges. This is called *exiting transfers* as opposed to *entering transfers*. We will see, that our choice is preferable due to performance advantages of the multi-criteria search.

2.3 Foot-Paths and Special Transfer Rules

We propose the following extensions to make the model fully realistic. In a real environment it is possible to walk from one station to another if the two

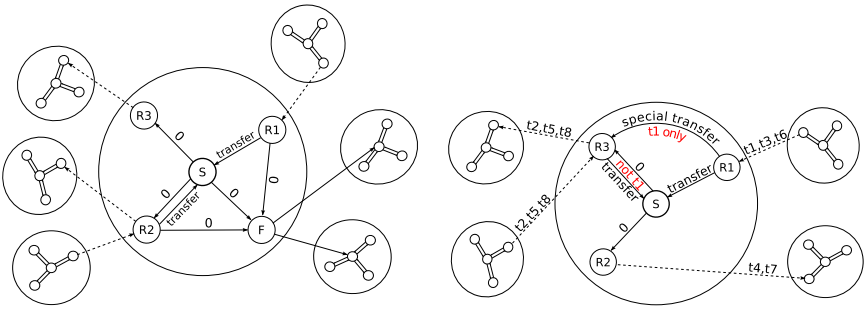

Fig. 2. (a) Illustration of a station with two foot-edges in the time-dependent model (b) Modifications to the graph for a station with a special transfer from train t_1 to train t_2

stations lie in geographic proximity. Realistic models therefore contain foot-paths to model this. Foot-paths are tuples (A, B, c) that represent a possibility to walk between stations A and B within c minutes. We assume, that c already contains all transfer costs at both A and B, so that no additional cost for switching trains arise. Foot-paths are special in that their length is constant in time. Figure 2 (a) shows the modifications that are needed in order to model a foot-path (A, B, c). It is not sufficient to simply add an edge from the station node of A to the station node of B with length c. This is because no additional transfer costs have to be paid when using a foot-path. Reducing c by the transfer cost at A, does not correctly model the costs when the journey starts at A. To circumvent these problems, an additional *foot-node* is added to the stations subgraph.

Another feature of realistic timetables are special transfer rules, that change the transfer time between two specific trains. The general transfer time of a station may be increased or decreased that way, depending on the real-world situation at the station. Two trains that use the same platform may for instance have a reduced transfer time. For each transfer rule several changes to the graph have to be made. Consider a special transfer time to get from train t_1 to train t_2 at station A. Let X denote the route node of A for t_1 and Y the route node of A for t_2. The station node for A is denoted by S. We assume that all special transfers are reasonable, i.e. it is not possible to reach a train departing before t_2 at Y if we arrived with t_1. However, there are cases in which it is explicitly made impossible to reach t_2 by setting the time of the special transfer higher than the usual transfer time. Figure 2 (b) shows the changes that have to be applied to the model when a special transfer rule is introduced. A new edge leads from X to Y carrying the special transfer cost. This edge may only be used after using t_1. The existing edges from S to Y and from Y to S have to be restricted so that they may *not* be used if t_1 is the last used train. This way Y cannot be reached from X without using the special transfer and the special transfer may not be used as shortcut to get to another route. For a proof of correctness of this model, we refer to the full version of this paper.

3 Multi-criteria Dijkstra and Speed-Up Techniques

In the following we briefly review an extension to Dijkstra's algorithm [16] that makes it capable of coping with several minimization criteria. Pseudocode is given in Algorithm 1. See Möhring [17] or the PhD-thesis of Theune [18] for a general description and correctness proofs.

3.1 Dijkstra's Algorithm for Multiple Criteria

The major difference when considering multiple search criteria is that nodes in the graph may be visited multiple times. The order in which nodes are visited in the classical algorithm guarantees that once a node is visited no better way of reaching it will be found later on. Multiple criteria allow for the possibility of different paths to a node that are not comparable – neither is strictly better.

Thus we need a way to remember all promising paths with which a node was reached. We do this by using multi-dimensional *labels*. Labels are associated with nodes and contain an entry for each criterion and a reference to its predecessor on the path. For every node in the graph we maintain a list of labels that are not dominated by any other label at the node. In the beginning, all label lists are empty. Then, start labels are created for all nodes with a timestamp within the query interval and are stored in a priority queue (lines 4-6). In the main loop of the algorithm, a lexicographically minimal label is extracted from the priority queue in each iteration (line 8). For the corresponding node of that label all outgoing edges are scanned and labels for their head nodes are created, provided that the edge is feasible (lines 9-11). Any new label is compared to all labels in the list corresponding to its node. It is only inserted into that list and into the priority queue if it is not dominated by any other label in the list. On the other hand, labels dominated by the new label are marked as invalid and removed from the node list (line 16).

3.2 Speed-Up Techniques

A good measure for the performance of this multi-criteria search algorithm is the number of labels created during a search. There are several techniques of reducing this number and thus increasing the algorithm's performance. As inserting labels into the priority queue is an expensive part of the search, the number of insertions can serve as a secondary measure of the algorithm's performance. We will now briefly discuss some important optimizations of the search, starting with some techniques that have been used in the time-expanded approach for some time [13]. In Section 3.2 we introduce two new and rather technical optimizations that have some impact on the search within the time-dependent graph.

Obtaining Lower Bounds. Some of the techniques described below make use of lower bounds for the distance of a node to the target node. These bounds can be available for some or all criteria. A general way of obtaining bounds is to

```
   Input: a timetable graph and a query
   Output: a set of Pareto-optimal labels at the terminal
 1 foreach node v do
 2     list<Label> labelListAt(v) := ∅;
 3 PriorityQueue pq := ∅;
 4 foreach node v in start interval do
 5     Label startLabel := createStartLabel(v);
 6     pq.insert(startLabel);
 7 while ! pq.isEmpty() do
 8     Label label := pq.extractLabel();
 9     foreach outgoing edge e=(v,w) of v=label.getNode() do
10         if isInfeasible(e) then continue; // ignore this edge
11         Label newLabel := createLabel(label, e);
12         if newLabel is dominated then continue;
13         // newLabel is not dominated
14         pq.insert(newLabel);
15         labelListAt(w).insert(newLabel);
16         labelListAt(w).removeLabelsDominatedBy(newLabel);
```

Algorithm 1. Pseudocode for the generalized Dijkstra algorithm

simplify the graph enough to make it possible to search quickly. In a simplified auxiliary graph, a single-criterion backward search is performed in order to obtain lower bounds for all nodes and one criterion. In order to be able to perform a backward search, any time-dependency must be eliminated.

We have implemented two different versions of simplified graphs with different properties. The more efficient one uses the graph of the basic time-dependent model in which only travel time can be optimized and transfers are costless. Time-dependency is removed by replacing variable edge costs with their minimal cost over time. This graph is suited for obtaining lower bounds for travel time only. Another simplification procedure keeps the complete graph and only substitutes time-dependent edges with constant ones as in the first approach. The resulting graph is more complicated but yields tighter bounds and can also be used for transfers.

Dominance by Early Results. The basic version of the generalized Dijkstra algorithm tests only labels which reside at the same node for mutual domination. Therefore, sub-optimality of sub-paths can often only be detected at a later stage — at the latest at the terminal. This causes a significant amount of wasted work which we try to avoid.

A way to improve the behavior is to explicitly check newly created labels against results we already have. If they are already worse they may be discarded right away. The sooner our first results are obtained, the less avoidable labels are created this way. Lower bounds may be used when trying to dominate a label

by earlier results. The criteria are therefore modified by adding the lower bound for the current node. That way labels that cannot lead to new Pareto optima are discarded as early as possible. If the lower bounds are tight enough, this can lead to major improvements once the first result has been found.

Goal-Directed Search. The lower bounds at a node can be used to add them as future costs to the cost values of a partial connection. If the extract operation from the priority queue is based on these modified values, this results in a goal-directed search as in the A* algorithm. It is important to note that this modification by itself leads to no improvement of running-time. It simply causes the search to find the first result earlier on. Together with the dominance by early results however, it leads to a major running-time improvement.

Avoid Hopping and Label Forwarding. Two phenomena that often arise when searching the time-dependent graph can be eliminated in order to improve performance. The first one is that labels propagate back to the node which they originated from. In this case the labels are immediately dominated. The search can easily be adapted to forbid *"hopping"*, i.e. the back-propagation of labels. The other phenomenon is due to the fact that all edges between station and route nodes in our graph have a cost of zero for all criteria. Because of this, newly created labels often have the same values for the single criteria as the label they originated from. Therefore, they are lexicographically minimal in the priority queue from the moment on they are inserted. We can thus avoid inserting them and simply hold them back until the current label has been processed completely. Before extracting further labels from the queue, the labels that are held back can be processed.

4 Computational Study

In the following, we analyze the performance of our multi-criteria search algorithm. We apply the above speed-up techniques and compare our prototype to a time-expanded approach. For the main part of our experiments we selected two relatively unrelated criteria, namely travel time and the number of transfers. Later we also show the influence on performance when adding an additional criterion to the search.

4.1 Train Network and Test Cases

The train network used in this study is derived from the train schedule of all trains within Germany of 2007 (56,994 trains, 8916 stations). The time-dependent graph has about 240,000 nodes and 670,000 edges while the corresponding time-expanded graph uses about 3,479,000 nodes and 5,633,000 edges. Three different sets of test cases were used. Each test case contains a source and a target station for the search, a date and a start time interval on that date. The first set of test cases is a synthetic one. It contains 1,000 randomly

created tests that allow for arbitrary start time intervals (referred to as *random cases*). The second set also contains 1,000 randomly created tests which however have more realistic start time intervals of exactly one hour (*realistic cases*). The third set contains about 14,000 tests that were obtained from a snapshot of real connection queries provided by Deutsche Bahn AG (*real cases*).

4.2 Computational Environment

All computations were executed on an AMD Athlon(tm) 64 X2 dual core processor 4600+ with 2.4 GHz and 4 GB main memory running under Suse Linux 10.2. Our C++ code has been compiled with g++ 4.1.2 and compile option -O3.

4.3 Experiments

We first analyze the impact of single speed-up techniques. As a main indicator for performance we use several operation counts on representative operations, most importantly on the number of created labels, as well as on the number of labels which pass the domination tests and are inserted into the priority queue. We also provide CPU times, however, since our system is just a prototype to demonstrate feasibility of the approach, no serious effort was spent on fine-tuning the code in order to improve running time directly.

Impact of Exact Speed-Up Techniques. We start with a *base-line variant* which is the generalized Dijkstra algorithm on the fully realistic graph model without using any optimization techniques and choosing exiting transfers (cf. Section 2.2). Our first investigation compares this base-line variant with an optimized version which includes domination by early results as well as goal direction. The lower bounds are obtained from the basic time-independent graph (cf. Section 3.2). In addition to that, avoidance of hopping and label forwarding are used. Table 1 shows the combined impact of these techniques on performance. We observe an improvement of a factor of about six with respect to the number of created labels and a factor of 13 with respect to the number of insertions into the priority queue. A more careful analysis reveals the individual impact of the low level optimizations of avoiding the hopping of labels and their forwarding along costless edges (cf. Section 3.2). This can be seen in Table 2.

It can also be seen that the choice between entering and exiting transfers (cf. Section 2.2) makes a notable difference in performance. Together a factor of nearly two is achieved in the number of created labels and a factor of over three is achieved in the number of inserted labels. Note that the running times of the different sets of queries cannot be compared. The real cases use start time intervals of three hours while the realistic cases use one hour. This leads to an average number of about six non-dominated solutions for the real cases, but only an average of about two for the realistic cases. Therefore, different running times are to be expected. Although the average number of created labels is similar for both sets of instances, the actual distribution of the number of created labels has a significantly larger variance for the real cases.

Table 1. Comparison of the base-line variant with an optimized version

	1000 realistic cases		
	created labels	inserted labels	average time in seconds
base-line variant	1236744	636393	4.730
optimized version	207976	47967	1.050

Table 2. Performance improvement when avoiding hopping and forwarding labels

	1000 random cases			1000 realistic cases			14076 real cases		
	created labels	inserted labels	avg. time	created labels	inserted labels	avg. time	created labels	inserted labels	avg. time
entering transfers	1232592	545416	7.049s	385982	160200	1.606s	386764	176540	2.360s
exiting transfers	1072187	552012	5.990s	315565	160516	1.311s	343248	177193	2.098s
avoid hopping	682925	552014	5.453s	207984	160514	1.183s	212503	177192	1.932s
avoid hopping + label forwarding	682897	146766	4.690s	207976	47967	1.050s	212516	45114	1.570s

As explained in Section 3.2, there are several ways of obtaining lower bounds. The above results used the basic time-dependent graph. However, by using the more complex approach, lower bounds can be obtained for other criteria as well, like the number of transfers. Unfortunately, the lower bounds on the number of transfers do not improve the search sufficiently to overcome the effort of determining the bounds in the first place, as can be seen in Table 3.

An improvement can still be achieved with tighter bounds on the travel time. Compared to not using any heuristic, we obtain an improvement of factor about two. The most efficient variant of these bounds — the complex graph with travel time bounds only — will from here on be used as our *standard variant* for further comparisons.

Further Speed-Up by Realistic Assumptions. One of the strengths of our approach is the guaranteed optimality of the search results. We are not willing to sacrifice this advantage by using speed-up techniques that violate optimality. The only exception are optimizations that use realistic assumptions in order to limit the search to certain reasonable ranges for the criteria. The results of applying some of these techniques are shown in the following. There are two ways of restricting the allowed travel time. Firstly it can be restricted by a fixed upper limit like 24 hours. This helps a lot for long connections but does not help at all for short ones. A more adaptive restriction is to limit the allowed travel time to γ times the time of the fastest connection, where γ is a variable parameter of our algorithm. This improves the search a lot for short queries. Our results are summarized in Table 4. To limit the number of transfers did not show a notable effect on performance in our tests. A maximum of five allowed transfers did not yield a better performance, even though it makes some

Table 3. Performance when using several combinations of the simple and the complex graph in order to obtain lower bounds (realistic cases)

heuristic for lower bounds	created labels	inserted labels	average time in seconds
none	420803	92305	1.839
simple (time)	207976	47967	1.050
complex (time)	205260	45886	1.003
simple (time), complex (transfers)	207813	47939	1.106
complex (time & transfers)	205101	45866	1.159

Pareto-optimal connections impossible. Hence we dropped the limit on the number of transfers completely. A reasonable limitation can be put on the maximum waiting time at a station since long waiting periods are very unattractive for most passengers. This especially improves the search for connections running over night. The improvement can be seen in Table 5. Finally, the single limits can be applied together in different ways. We applied *conservative limits* of 24 hours for maximum travel time, five hours for maximum waiting time and $\gamma = 5$ and *tight limits* of ten hours for maximum travel time, three hours for maximum waiting time and $\gamma = 2$. The improvements can be found in Table 6. In summary, together with the exact speed-up techniques, a speed-up factor of about 20 over the base-line version has been achieved with respect to the number of created labels and a factor of 138 with respect to the number of insertions.

Comparison with a Time-Expanded Approach. In general, we expect a better performance of the time-dependent approach than of the time-expanded one. It is unclear however, whether this can be achieved in a multi-criteria setting. In order to answer this question, we compare the performance of our time-dependent approach with the time-expanded search incorporated in MOTIS. As the time-dependent system was developed as a proof of concept only, it makes not much sense to compare running times. We restrict our analysis to the comparison of the number of labels inserted into the priority queue. As can be seen in Table 7 the time-dependent approach creates much fewer labels. When using realistic assumptions, the time-dependent system adds 5.4 times less labels into the priority queue. However, it should be noted, that the time-dependent approach requires additional effort to compute actual edge lengths on-the-fly. Thus we expect (and empirically observe) similar running times for both approaches. As expected, the memory consumption of the time-expanded graph is a lot higher than that of the time-dependent one. In our tests, MOTIS needed 1.25 GB while the time-dependent graph used only 281 MB.

Adding an Additional Criterion: Reliability of Transfers. The above experiments were performed using travel time and the number of transfers as only search criteria. An interesting question is how the performance worsens when further criteria are introduced. This was explored by adding the "reliability of transfers" as a further criterion.

Table 4. Limiting the maximum travel time (realistic cases)

algorithmic variant	created labels	inserted labels	average time in seconds
standard	205260	45886	1.003
max. travel time = 24h	180910	30893	0.845
max. travel time = 15h	141602	16175	0.631
max. travel time = 10h	83162	6999	0.406
$\gamma = 5$	182535	32030	0.865
$\gamma = 3$	144678	17015	0.653
$\gamma = 2$	84125	6890	0.415

Table 5. Limiting the maximum waiting time (realistic cases)

algorithmic variant	created labels	inserted labels	average time in seconds
standard	205260	45886	1.003
max. waiting time = 5h	167914	19751	0.777
max. waiting time = 3h	151680	15441	0.637

Table 6. Performance improvement when combining limits (realistic cases)

algorithmic variant	created labels	inserted inserted	average time in seconds
standard	205260	45886	1.003
conservative limits	156515	17827	0.685
tight limits	63261	4605	0.335

The reliability of a single transfer is a function of the *buffer time* which is the available time exceeding the minimum transfer time at the station. This means that a passenger will catch the connecting train unless the incoming train is delayed by more than the buffer time. There are many plausible ways to map a buffer time t into a reliability measure. In this paper, we propose to define

$$\text{reliability} : t \mapsto s - \exp^{\ln(1-a)-\frac{1}{b}\cdot t},$$

with parameters $a = 0.6$, $b = 8$, $s = 0.99$ so that the maximal reliability of a single transfer is 99% and a buffer time of 0 minutes leads to 60% reliability. The reliability of connections with several transfers is defined as the product of the reliabilities of each single transfer. This yields a continuous reliability measure which we further transformed into a discrete one by subdividing the interval of [0,1] into 50, 20 and 10 equivalence classes of equal width. Table 8 summarizes the performance of the search when using different numbers of criteria. The addition of the number of transfers as second criterion leads to a slow-down of

Table 7. The number of labels inserted into the priority queue on average for both the time-dependent and the time-expanded search

real cases	inserted labels
time-expanded (optimized, conservative limits)	92538
time-expanded (optimized, tight limits)	64782
time-dependent (optimal, no limits)	44133
time-dependent (realistic assumptions, tight limits)	11913

Table 8. Relationship between the number of criteria and performance on 1000 realistic test cases. Different numbers of discretization steps are used for reliability.

criteria	created labels	inserted labels	average time in seconds	average number of Pareto optima
time	99284	19401	0.454	1.28
time, transfers	205260	45886	1.003	2.34
time, transfers, reliability (50 classes)	990664	160254	5.726	6.76
time, transfers, reliability (20 classes)	853742	149366	4.727	5.67
time, transfers, reliability (10 classes)	772822	142615	4.138	4.66

factor two, the addition of reliability of transfers as third criterion leads to a slow-down of another factor four if we use 10 equivalence classes.

5 Conclusions and Future Work

In this work we have presented our prototype for a time-dependent, multi-criteria search system that works in a fully realistic scenario. We have shown how to introduce the most important features of real timetables and how to improve performance significantly. We have provided the results of our experimental analysis that show that a speed-up factor of 20 with respect to the number of label creations and 138 with respect to the number of label insertions can be achieved under realistic assumptions. A comparison to the time-expanded approach was done, indicating that the new approach clearly is competitive. Finally we discussed the impact on performance when adding further criteria to the search.

In order to make the time-dependent approach able to replace current online search systems, its performance needs to be improved further. If possible, optimality should be maintained. It remains a challenge to design better speed-up techniques for multi-criteria search. Another goal is to extend our prototype to a dynamic scenario with train delays.

Acknowledgments

This work was partially supported by the DFG Focus Program Algorithm Engineering, grant Mu 1482/4-1. We wish to thank Deutsche Bahn AG for providing us timetable data for scientific use.

References

1. HaCon web-site (2007), http://www.hacon.de/hafas/konzept.shtml
2. Müller-Hannemann, M., Weihe, K.: On the cardinality of the Pareto set in bicriteria shortest path problems. Annals of Operations Research 147, 269–286 (2006)
3. Pallottino, S., Scutellà, M.G.: Shortest path algorithms in transportation models: Classical and innovative aspects. In: Equilibrium and Advanced Transportation Modelling, Kluwer Academic Publishers, Dordrecht (1998)
4. Schulz, F., Wagner, D., Weihe, K.: Dijkstra's algorithm on-line: An empirical case study from public railroad transport. ACM Journal of Experimental Algorithmics, Article 12, 5 (2000)
5. Pyrga, E., Schulz, F., Wagner, D., Zaroliagis, C.: Efficient models for timetable information in public transportation systems. ACM Journal of Experimental Algorithmics (JEA) 12, 2.4 (2007)
6. Cooke, K.L., Halsey, E.: The shortest route through a network with time-dependent internodal transit times. Journal of Mathematical Analysis and Applications 14, 493–498 (1966)
7. Orda, A., Rom, R.: Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. Journal of the ACM 37, 607–625 (1990)
8. Orda, A., Rom, R.: Minimum weight paths in time-dependent networks. Networks 21, 295–319 (1991)
9. Nachtigal, K.: Time depending shortest-path problems with applications to railway networks. European Journal of Operations Research 83, 154–166 (1995)
10. Brodal, G.S., Jacob, R.: Time-dependent networks as models to achieve fast exact time-table queries. In: Proceedings of the 3rd Workshop on Algorithmic Methods and Models for Optimization of Railways (ATMOS 2003). Electronic Notes in Theoretical Computer Science, vol. 92, pp. 3–15. Elsevier, Amsterdam (2004)
11. Pyrga, E., Schulz, F., Wagner, D., Zaroliagis, C.: Towards realistic modeling of time-table information through the time-dependent approach. In: Proceedings of the 3rd Workshop on Algorithmic Methods and Models for Optimization of Railways (ATMOS 2003). Electronic Notes in Theoretical Computer Science, vol. 92, pp. 85–103. Elsevier, Amsterdam (2004)
12. Müller-Hannemann, M., Schulz, F., Wagner, D., Zaroliagis, C.: Timetable information: Models and algorithms. In: Geraets, F., Kroon, L.G., Schoebel, A., Wagner, D., Zaroliagis, C.D. (eds.) Railway Optimization 2004. LNCS, vol. 4359, pp. 67–89. Springer, Heidelberg (2007)
13. Müller-Hannemann, M., Schnee, M.: Finding all attractive train connections by multi-criteria Pareto search. In: Geraets, F., Kroon, L.G., Schoebel, A., Wagner, D., Zaroliagis, C.D. (eds.) Railway Optimization 2004. LNCS, vol. 4359, pp. 246–263. Springer, Heidelberg (2007)
14. Bauer, R., Delling, D., Wagner, D.: Experimental study on speed-up techniques for timetable information systems. In: ATMOS 2007 (2007)
15. Bauer, R., Delling, D.: SHARC: Fast and Robust Unidirectional Routing. In: Proceedings of the 9th Workshop on Algorithm Engineering and Experiments (ALENEX 2008) (2008)
16. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik 1, 269–271 (1959)
17. Möhring, R.H.: Verteilte Verbindungssuche im öffentlichen Personenverkehr: Graphentheoretische Modelle und Algorithmen. In: Angewandte Mathematik - insbesondere Informatik, Vieweg, pp. 192–220 (1999)
18. Theune, D.: Robuste und effiziente Methoden zur Lösung von Wegproblemen. Teubner Verlag, Stuttgart (1995)