Accelerating Time-Dependent Multi-Criteria Timetable Information is Harder Than Expected*

Annabell Berger[1], Daniel Delling[2], Andreas Gebhardt[1], and Matthias Müller-Hannemann[1]

[1] Department of Computer Science, Martin-Luther-University Halle-Wittenberg, Von-Seckendorff-Platz 1, 06120 Halle, Germany
{berger,gebhardt,muellerh}@informatik.uni-halle.de
[2] Department of Computer Science, University of Karlsruhe, P.O. Box 6980, 76128 Karlsruhe, Germany. delling@informatik.uni-karlsruhe.de

Abstract. Speeding up multi-criteria search in real timetable information systems remains a challenge in spite of impressive progress achieved in recent years for related problems in road networks. Our goal is to perform multi-criteria range queries, that is, to find all Pareto-optimal connections with respect to travel time and number of transfers within a given start time interval. This problem can be modeled as a path search problem in a time- and event-dependent graph. In this paper, we investigate two key speed-up techniques for a multi-criteria variant of Dijkstra's algorithm — arc flags and contraction — which seem to be strong candidates for railway networks, too. We describe in detail how these two techniques have to be adapted for a multi-criteria scenario and explain why we can expect only marginal speed-ups (compared to observations in road networks) from a direct implementation. Based on these insights we extend traditional arc-flags to *time-period flags* and introduce *route contraction* as a substitute for node contraction. A computational study on real queries demonstrates that these techniques combined with goal-directed search lead to a speed-up of factor 13.08 over the baseline variant for range queries for a full day.

Keywords: timetable information, multi-criteria search, time-dependent networks, arc flags, contraction

1 Introduction

In recent years there has been growing interest in high-performance timetable information systems [22]. While exact single-criterion search is well understood and already quite efficient, multi-criteria timetable information remains a challenge. Therefore, commercial state-of-the-art systems still use only heuristics to

* 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.

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/2009/2148

determine relevant connections for their customers. Since there has been impressive progress with speed-up techniques for related problems in road networks, it seems natural to start an attempt to transfer the underlying methods to a railway scenario.

In this paper, we report on a project where we worked out the necessary details to augment standard search techniques by additional information obtained in a preprocessing phase. We investigate two key speed-up techniques for a multi-criteria variant of DIJKSTRA's algorithm — arc flags and contraction.

Related Work. Many speed-up techniques for single-criteria scenarios have been developed during the last years. Due to space limitations, we direct the interested reader to [8] and [10], which give recent overviews over single-criteria time-independent and time-dependent route planning techniques, respectively.

Basics. The straightforward approach to find all Pareto optimal paths is the generalization [15, 18, 20] of DIJKSTRA's algorithm: Each node $v \in V$ gets a number of multi-dimensional labels assigned, representing all Pareto paths to v. For the bicriteria case, Hansen [15] was the first presenting such a generalization, while Theune [30] describes multi-criteria algorithms in detail. By this generalization, DIJKSTRA loses the property that each node is visited only once. It turns out that a crucial problem for multi-criteria routing is the number of label entries assigned to the nodes. The more label entries are created, the more nodes are reinserted in the priority queue yielding considerably slow-downs compared to the single-criterion setup. In the worst case, the number of labels can be exponential in $|V|$ yielding impractical running times [15], and also memory consumption becomes an issue. Hence, Hansen [15] and Warburtun [31] present an FPTAS (fully polynomial time approximation scheme) for the bicriteria shortest path problem.

Speed-up Techniques. Most of the work on speed-up techniques for multi-criteria scenarios was done on networks derived from timetable information. In such networks, Müller-Hannemann and Weihe [23] observed that the number of labels is often limited such that the brute force approach for finding *all* Pareto paths is often feasible. Experimental studies finding Pareto paths in timetable graphs can be found in [25, 26, 29, 27, 21, 14, 11]. We would like to point out that one has to distinguish between finding all Pareto paths and only finding one representative for each equivalenve class of paths with the same tuple of objective values. Previous work usually guarantees only the weaker second version.

SHARC, a route planning algorithm developed by one of this work's coauthors, has been introduced in [2, 3]. Originally, SHARC only worked on time-independent networks. In [6, 7], it has successfully been adapted to time-dependent road and railway networks, and very recently, even to a (time-independent) multi-criteria scenario [9]. However, experiments for the multi-criteria variant were only conducted on time-independent road networks. So, to the best of our knowledge, no advanced speed-up technique has yet been adapted to a realistic multi-criteria timetable information system on time-dependent networks.

Our contribution and overview. This paper is devoted to transfer advanced speed-up techniques to time-dependent railway networks. In contrast to most previous scientific work, we consider a scenario with the following features:

- Our model is a *fully realistic model*, where traffic days, business rules on required transfer times between connecting trains, footpaths between neighboring stations, train attributes, and the like are respected.
- We aim at finding *all Pareto optimal paths* for two criteria, travel time and number of transfers. We would like to emphasize that we here mean the strong version which really enumerates all Pareto paths, and not just one representative path for each non-dominated pair of objective values. Since there are often several possibilities to change between the same two trains, this leads to a much larger set of paths. The motivation to search for these paths comes from practice: railway companies have preferences at which stations their passengers should change trains. Hence, they would like to select from the complete set of Pareto paths a subset which they present to customers.
- We want to perform a *range search* for an arbitrary user-specified start-time interval (not only from a single desired start point). As a result, we are able to compute the complete connection table between two arbitrary stations for a full day.

To model this scenario we will introduce a station graph model with train routes which is slightly more compact than those used in Disser et al. [11]. While Dijkstra's algorithm can be easily generalized to time-dependent graphs in the single-criterion case [5], one has to be more careful in a multi-criteria setting. The crucial operation in a multi-criteria search algorithm is to decide which subpaths can be safely dominated. To ensure correctness subpath optimality is required, and therefore Müller-Hannemann and Berger [4] extended the classical time-dependent model to an event-dependent model.

In this work, we mainly investigate two prominent speed-up techniques, arc-flags and contraction, and their combination. We

- discuss how these techniques have to be adapted to work for the above scenario,
- explain why they do not lead to as large speed-ups as one might have hoped for, and
- develop two new refinements which achieve at least some significant speed-up over previous work on range queries.

Classical arc flags turn out to be rather weak for arbitrary multi-criteria range queries: almost all arc flags must be set to `true` to guarantee correctness of the query algorithm since for any arc there is almost surely one point in time where this arc is part of some Pareto-optimal path towards the target station. However, from our preprocessing we do know exactly at which points of time any particular arc might be necessary. By this observation we refine the classical arc flags to *time-period arc flags*. The idea is to divide the overall range for which

our preprocessing is valid into short time intervals, for example into intervals of two hours. Then each arc maintains a flag for each combination of time interval (*period*) and region which tells whether the arc might be "useful" for a particular query.

Standard node contraction suffers from the dilemma that our station graph has due to many parallel routes already a very high average degree of ≈ 43 (in comparison, road networks have empirically an average degree below 4). Thus, bypassing a node leads to the introduction of many shortcut arcs. While many shortcut arcs can be pruned away in a single-criterion search in time-independent road networks, domination criteria in a multi-criteria scenario are much weaker in event-dependent railway networks, as we will explain in Section 4. Therefore, we decided to develop and implement a different concept which can be combined with arc-flags: *route contraction*. The idea behind route contraction is to insert for a path composed by arcs on the same route a new shortcut arc, provided that all intermediate stations on this path are classified as bypassable. A station is *bypassable* if (a) it is neither the beginning or end of some route, (b) it has at most two different neighbors, and (c) it is not a boundary node of some region used in the node partition for the arc-flags. In Germany, about 60% of all stations are bypassable with respect to this definition.

In addition, we have realized a variant of goal-directed search which for each query first computes minimum travel times from each node towards the target station and then uses these values as lower bounds during the search. Extensive computational experiments indicate that the combination of these methods together with a greedy strategy allow range queries for a full day in about 0.53 seconds. This gives a speed-up of about 10.1 over our baseline variant.

The remainder of the paper is organized as follows. In Section 2, we briefly review the classical arc-flag method and SHARC. Then, in Section 3, we discuss modeling issues for multi-criteria time-table information. We introduce our station graph model and explain the baseline variant of a multi-criteria generalization of Dijkstra's algorithm. Afterwards, we describe how to adapt the preprocessing phase for arc-flags and contraction to a multi-criteria time-dependent version. In particular, we introduce the new concepts of time-period arc-flags and route contraction. Results of an experimental study are presented in Section 5. Finally, we conclude with a short summary.

2 Preliminaries

A (directed) graph $G = (V, A)$ consists of a finite set V of nodes and a finite set A of *arcs*. An arc is an ordered pair (u, v) of nodes $u, v \in V$, the node u is called the *tail* of the arc, v the *head*. Throughout the whole work we restrict ourselves to directed graphs which are weighted by a length function *len*, which we specify in Section 3. A *partition* of V is a family $\mathcal{C} = \{C_0, C_1, \ldots, C_k\}$ of sets $C_i \subseteq V$ such that each node $v \in V$ is contained in exactly one set C_i. An element of a partition is called a *region*. The *boundary nodes* B_C of a region C are all nodes

$u \in C$ for which at least one node $v \in V \setminus C$ exists such that $(v, u) \in A$. We call v a *pre-boundary* node of the region u is assigned to.

SHARC. Introduced in [2, 3], SHARC combines ideas from arc-flags [17, 16] and contraction [28, 12]. The original arc-flag approach first computes a partition $\mathcal{C}$ of the graph and then attaches a *label* to each arc a. A label contains, for each region $C \in \mathcal{C}$, a flag $AF_C(a)$ which is true if a shortest path to at least one node in C starts with a. A modified DIJKSTRA then only considers those arcs for which the flag of the target node's region is true. The main downside of this approach is the high preprocessing effort. Hence, SHARC improves on this by the integration of contraction, i.e., a routine iteravely removing unimportant nodes and adding so-called *shortcuts* in order to preserve distances between non-removed nodes. One key observation of SHARC is that we are able to assign arc-flags to all bypassed arcs during contraction. More precisely, any arc (u, v) outgoing from a non-removed node and heading to a removed one gets only one flag set to true, namely, for the region v is assigned to. Any other bypassed arc gets all flags set to true. By this procedure, unimportant arcs are only relaxed at the beginning and end of a query.

3 Modeling Issues

Up to now, two models have been introduced for efficient timetable information systems: the time-expanded and time-dependent approach. See the survey paper [22] for details. In this section we extend the time-dependent approach to an *event-dependent* scenario (see [4]) and introduce a more compact graph model.

3.1 Elementary Connections, Connections and Connection Tables.

Before explaining our station graph model, we need the notion of connections within a timetable. Let S be the set of stations. An *elementary connection* $c_e = (dep_v(time), arr_w(time), T)$ represents exactly one train T which departs at time $dep_v(time)$ in station $v \in S$ and arrives at arrival time $arr_w(time)$ in station $w \in S$ without stops. An *elementary connection-table* C_e is a set of elementary connections with identical origin v and destination w. Furthermore, there exists a set of minimum transfer times $trans_s(T, T') \in \mathbb{N}$ between trains T, T' with respect to each station $s \in S$. These transfer times ensure the possibility to transfer between two trains with respect to different situations. We call two elementary connections $c_e = (dep_v(time), arr_w(time), T)$ and $c'_e = (dep_{v'}(time), arr_{w'}(time), T')$ *concatinable* if and only if $w = v'$ and $dep_{v'}(time) - arr_w(time) \geq trans_w(T, T')$. We denote a sequence of elementary connections $c_{e_1}, \ldots, c_{e_k}$ as *connection* $c = (c_{e_1}, c_{e_k}, transfer)$ if each adjacent pair of elementary connections $(c_{e_i}, c_{e_{i+1}})$ in the sequence is concatinable. Attribute *transfer* counts the number of transfers using connection c. Note, that this definition allows to concatenate connections if there ending and starting elementary connections are concatinable. We denote with $c(dep_v(time))$, $c(dep_v(train))$

and $c(arr_w(time))$, $c(arr_w(train))$ the starting and ending departure and arrival times/trains of connection c. Analogously to elementary connection-tables we define a *connection-table* C as a set of connections with identical origin v and destination w. Last, we define an operator $\oplus$ on connection tables C, C' which assigns to each pair of connection tables (C, C') a new connection table C''. C'' contains all connections c'' consisting of concatinable pairs of connections $(c, c') \in C \times C'$. In the following, we assign elementary connection-tables to arcs but also compute connection-tables between arbitrary pairs of stations.

3.2 Station Graph Model

Our approach is based on a directed graph $G = (V, A)$ without loops but with parallel arcs which is called *station graph*. Each node $v \in V$ models a station $s \in S$. Inserting arcs is more sophisticated. In a first step we connect two stations if and only if there exist at least one elementary connection between these stations. Next, we identify trains with the following properties: they stop exactly at the same sequence of stations, have the same train attributes and days of operation, and never violate the FIFO property, i.e., they always run in the same order on each arc. We denote such sequences of stations as *routes* and get for each arc a set of different routes using this arc. Now, we replace each arc (v, w) by parallel *route arcs* $(v, w)_i$, one for each route on this arc. We add the new attribute *route number* to each elementary connection. In a last step we assign to each route arc the corresponding elementary connection-table.

Foot-Arcs. Our data also contains foot paths modeling inter-station transfers reachable by foot. In our graph model, we simply connect the corresponding stations v, w by a foot-arc with constant length l corresponding to the time necessary for traversing the arc (v, w) by foot F. Hence, we can associate with each foot arc an elementary connection table which contains for each discrete point of time an elementary connection $c_e = (dep_v(time), arr_w(time), F)$ with $arr_w(time) - dep_v(time) = l$.

3.3 Route Planning in the Station Graph Model

In this work, we concentrate on computing optimal connection tables between two arbitrary stations s and t at a given start time interval $[\tau_{start}, \tau_{end}]$ for station s with respect to the travel time and number of transfers. We denote the travel time of a connection c with $ttime(c)$ and the number of transfers with $transfer(c)$. Each connection can be seen as an event-dependent path in the station graph. Müller-Hannemann and Berger introduced *event-dependent models* as an extension of time-dependent approaches in [4]. The reason to introduce this extension is that our second optimization criterion "number of transfers" not only depends on time but additionally on train numbers. This leads to new definitions for time-dependent settings and their generalizations. First, we assign to each arc $a = (v, u) \in A$ and departure event dep_v at v an arrival event

arr_u which defines the arrival event at vertex u if we depart in v with departure event dep_v and traverse arc a. This models our elementary connections. For time-dependent models an event consists only of the attribute time. Therefore, all departure events with the same departure time at a vertex v will be considered as equal events. In our scenario an event consists of attributes departure or arrival time, train number and route number. We define for all $v \in V$ a set of departure events Dep_v and arrival events Arr_v. Consider all connections in a connection table between station s and t. Then such a connection $c = ((dep_s(time), arr_w(time), T), (dep_v(time), arr_t(time), T'), transfer)$ is an alternating sequence $(dep_s, arr_w, \ldots, dep_v, arr_t)$ of departure and arrival events which consist of attributes $(time, train, routenumber)$. For an (s,t)-query we ignore all arrival events at s, but add an artificial "arrival event" $start_s$ with an earliest start time $start_s(time) := \tau_{start}$ at the beginning of c. Furthermore, we define one artificial "departure event" end_t which is added to the end of c. We denote such an alternating sequence as *event-dependent path* $P_{start_s,end_t} := (start_s, c, end_t)$. Furthermore, we call an alternating subsequence of an event-dependent path P_{start_s,end_t} starting at $start_s$ and ending in an arrival event arr_v as event-dependent subpath P_{start_s,arr_v}. We define the weight $w(P_{start_s,arr_v}) \in \mathbb{N}^2$ of an event-dependent path P_{start_s,arr_v} in the first component as the travel time $ttime(c)$ and in the second component as the number of transfers $transfer(c)$ of the underlying connection c. Note that all events belonging to an event-dependent path are distinct, but we do not rule out that corresponding stations are repeated.

If we want to use a generalized version of DIJKSTRA's algorithm to compute all event-dependent Pareto-paths, we need for correctness subpath optimality. To decide the optimality of an event-dependent subpath we may only compare subpaths which possess on their ends identical departure events, see [4]. Hence, in the case of a time-dependent scenario we may compare all subpaths which possess on their ends only identical arrival times. A generalized version of DIJKSTRA's algorithm, (see Algorithm 1), computes all event-dependent Pareto-paths. This algorithm uses a data structure for a label L which consists of

1. an arrival event arr_v,
2. a list l_w of *weights* $w \in \mathbb{R}_+^k$ for event-dependent paths P_{start_s,arr_v},
3. a list l_p of predecessor arrival events arr_u for event-dependent paths P_{start_s,arr_v}.

Note that in this version we construct a label for each route arc and this notion of a label includes all partial connections from the start station. Thus, we can identify such a label with a computed connection table representing all non-dominated connections from the start station up to the corresponding arc found so far. Upon termination, each label includes all Pareto-optimal paths.

To decide whether two alternatives dominate each other or not, we are able to compare all event-dependent subpaths not only ending with identical departure events but ending with different departure events and an identical route number. Hence, we can give special rules to delete some of these subpaths. In the next section we explain these "rules of dominance".

Algorithm 1: Generalized Dijkstra Event-Dependent

Input: Origin s, destination t, earliest start time $\texttt{start}_s(\texttt{time})$
Output: Set of all event-dependent Pareto-optimal (s, t)-paths.

```
1  create empty priority pq;
2  for arrival events arr_v do
3      if v ≠ s then construct label L_{arr_v} with empty list l_w;
4      else
5          construct label L_{start_s};
6          pq.insert(L_{start_s});
7  while ¬pq.empty() do
8      L_{arr_v} ← pq.extract-min() /* key is the smallest arrival time
           arr_v(time)                                                  */
9      compute with respect to trans_v possible departure events dep_v at vertex v;
       /* each departure event belongs to exactly one arrival event    */
10     determine the corresponding arrival event arr_u to L_{arr_u};
11     for these arrival events arr_u do
12         if label L_{arr_u} ∉ pq then pq.insert(L_{arr_u}) and store a flag that L_{arr_u} is
           in pq;
13         for weights stored in L_{arr_v}.l_w do
14             w(P_{start_s, arr_u}) ← w(P_{start_s, arr_v}) + w(arr_v, dep_v) + w(dep_v, arr_u);
15             if w(P_{start_s, arr_u}) not dominated in L_{arr_u}.l_w then
               L_{arr_u}.l_w.insert(w(P_{start_s, arr_u}));
16     delete dominated weights in label L_{arr_u}.l_w;
```

Rules of Dominance. Our station graph model allows additional rules to compare connections within each connection table on a route arc. In general, we may only compare connections with identical ending arrival times in one connection table. In our scenario the rules of dominance with respect to subpath-optimality don't change but in several cases we can decide the non-optimality of some subpaths in advance. Consider the computed connection table on route arc r in Figure 1. The third connection will be deleted because there is no Pareto-optimal (s,t)-path which can contain this connection as a subpath. Assume, this would be the case. Then the first connection in our time table can use the same connection from v to t as in this Pareto-path. Because the first and third connection end on the same route arc either both have to transfer at v or both continue on the same route. Hence, the (s,t)-path using connection 1 possess a smaller travel time and a smaller number of transfers. In contradiction to our assumption the path using connection 3 is dominated. Note, that we cannot delete connection 2 in this connection table. If the last train of connection 2 is the same as the only elementary connection on (v,t), connection 2 can be extended to a Pareto-optimal path from (s,t). Similar but stronger arguments can be found in comparing connection tables of two different route arcs r, r' ending at station v. In Table 1 we give our special deletion rules. We call the rules in line 1 and 2 *route dominance* and the rule in line 3 *station dominance*.

	c_1, c_2 comparable if	delete $c_2 \Leftrightarrow$
1	$c_1(arr_v(time)) \leq c_2(arr_v(time))$ $c_1(arr_v(route)) = c_2(arr_v(route))$	$c_1(arr_v(time)) - ttime(c_1) \geq$ $c_2(arr_v(time)) - ttime(c_2)$ $transfer(c_1) < transfer(c_2)$
2	$c_1(arr_v(time)) \leq c_2(arr_v(time))$ $c_1(arr_v(route)) = c_2(arr_v(route))$	$c_1(arr_v(time)) - ttime(c_1) >$ $c_2(arr_v(time)) - ttime(c_2)$ $transfer(c_1) \leq transfer(c_2)$
3	$c_1(arr_v(time)) \leq c_2(arr_v(time))$	$c_1(arr_v(time)) - ttime(c_1) >$ $c_2(arr_v(time)) - ttime(c_2)$ $transfer(c_1) < transfer(c_2)$

Table 1. Comparability and deletion criteria of two connections on route arcs ending in station v.

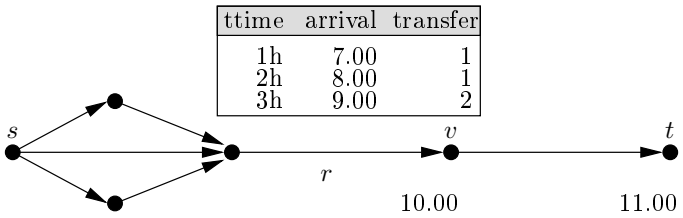

Fig. 1. Example: Route dominance at a connection table for paths from s to arc r. Note that we cannot delete connection 2 in this table if the elementary connection on arc (v, t) uses the same train as connection 2. However, connection 3 can be safely deleted.

4 Augmenting Ingredients

In this section, we present how to adapt the basic contraction and arc-flags to our scenario.

4.1 Contraction

One of the main reasons of the success of recent hierarchical (single-criteria) speed-up techniques is contraction, a routine that iteratively removes unimportant nodes from the graph and inserts so called shortcuts to preserve correct distances between the remaining nodes. Hence, in order to use this technique in our scenario, we need to augment this concept. In general, contraction works in two phases: vertex- and arc-reduction.

Vertex-Reduction. Adaption of vertex-reduction is straightforward. We *bypass* a node u by removing all its incoming arcs $I(u)$ and all outgoing arcs $O(u)$. In order to preserve Pareto-paths between the remaining nodes, we introduce, for each combination $(v, u) \in I(u)$, $(u, v') \in O(u)$ and their connection tables $C_{(v,u)}$ and $C_{(u,v')}$, a new arc (v, v') with connection-table $C_{(v,v')} = C_{(v,u)} \oplus C_{(u,v')}$.

	c_1, c_2 comparable if	delete $c_2 \Leftrightarrow$
1	$c_1(dep_v(event)) = c_2(dep_v(event))$ $c_1(arr_w(route)) = c_2(arr_w(route))$	$ttime(c_1) \leq ttime(c_2)$ $transfer(c_1) < transfer(c_2)$
2	$c_1(arr_w(event)) = c_2(arr_w(event))$ $c_1(dep_v(route)) = c_2(dep_v(route))$	$ttime(c_1) \leq ttime(c_2)$ $transfer(c_1) < transfer(c_2)$
3	$c_1(dep_v(event)) = c_2(dep_v(event))$	$ttime(c_1) \leq ttime(c_2)$ $transfer(c_1) + 1 < transfer(c_2)$
4	$c_1(arr_w(event)) = c_2(arr_w(event))$	$ttime(c_1) \leq ttime(c_2)$ $transfer(c_1) + 1 < transfer(c_2)$
5		$c_1(dep_v(time)) > c_2(dep_v(time))$ $ttime(c_1) + c_1(dep_v(time)) \leq$ $c_2(dep_v(time)) + ttime(c_2)$ $transfer(c_1) + 2 < transfer(c_2)$

Table 2. Comparability and deletion criteria of two connections on parallel shortcut arcs (u, v).

From vertex-reduction in other scenarios, we know that the order in which we remove vertices from the graph changes the resulting graph. Hence, we use a priority queue to determine which node to bypass next. The priority of a node u within the queue is defined by the expansion $\zeta(u) := (\deg_{in}(u) \cdot \deg_{out}(u))/(\deg_{in}(u) + \deg_{out}(u))$. We stop the vertex-reduction as soon as we would bypass a node with an expansion beyond a given threshold. All nodes remaining in the graph, we call core-nodes. The core of a graph contains all core-nodes and all arcs (including shortcuts) between core-nodes.

Theorem 1. *Vertex-reduction preserves event-dependent Pareto-optimal paths between core-nodes.*

Arc-Reduction. Our vertex-reduction creates a new connection-table for each added shortcut yielding quite a high increase in the total number of connections in the graph. Fortunately, we can remove some connections on the shortcuts because they may be dominated by other connections. In the best case, all connections on a shortcut are dominated. Then, we can safely remove the shortcut from the graph. One might expect that it sufficient to run a $(v$–$v')$–query for each added shortcut (v, v') and then remove all connections from (v, v') that are dominated. Unfortunately, this violates correctness since (v, v') can be a suffix and/or prefix of a shortest path (cf. Section 3). Still we can run a $(v$–$v')$–query for each shortcut but in order to preserve correctness, we have to use weaker (than those introduced in Section 3) rules of dominance during the query. These weaker rules are given in Table 2. The reason for these modified rules is that we have to compare paths ending in possibly two different events.

Theorem 2. *Arc-Reduction preserves event-dependent Pareto-optimal paths between core-nodes.*

The proof of Theorem 2 can be found in Appendix A. In Figures 2-4, we give an example how Vertex-Reduction and Arc-Reduction work in our scenario.

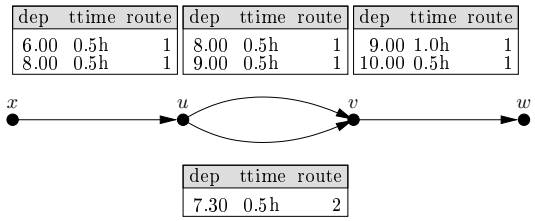

Fig. 2. Small excerpt of the station graph with elementary connections.

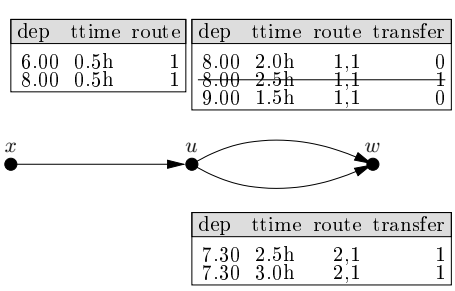

Fig. 3. Vertex-reduction at vertex v.

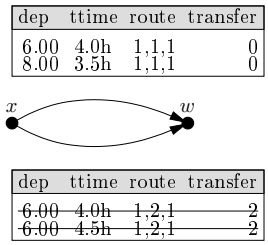

Fig. 4. After vertex-reduction at u, an arc-reduction of the lower arc between x and w is possible.

Figure 2 represents a small excerpt of a station graph with elementary connection tables on each route arc. In Figure 3, we delete vertex v and determine new connection tables on short cut arcs. Note, that none of the new connection tables can be deleted. In Figure 4, vertex u is deleted and the new connection table on the lower arc (x, w) is dominated and can be deleted.

Route Contraction. As mentioned in the Introduction, this standard node contraction suffers from the dilemma that our station graph has already a very high average degree of ≈ 43 due to the many parallel routes (in comparison, road networks have empirically an average degree below 4). Thus, bypassing a node leads to the introduction of many shortcut arcs which cannot be deleted. Therefore, we decided to develop and implement a different concept: *route contraction*. In a first step we partition the set of stations S in k several subsets C_i with $i \in \{1, \ldots, k\}$ which we call *regions*. The idea behind route contraction is to insert for a path composed by arcs on the same route a new shortcut arc, provided that all intermediate stations on this path are classified as bypassable. Recall from the Introduction that a station is *bypassable* if (a) it is neither the beginning or end of some route, (b) it has at most two different neighbors, and (c) is not boundary node of some region C_i. Thus our notion of bypassable nodes models in some sense "unimportant stations", for which we assume that at them no transfer makes sense. In Germany, about 60% of all stations are bypassable with respect to this definition. After determining all bypassable vertices in station graph G we can identify inclusion-maximal paths $P_{v,w}$ from v to w

containing only arcs of the same route and only bypassable vertices $u \neq v, w$ in its interior. Each such path $P_{v,w}$ is contracted to a *shortcut arc* (v, w). Arc (v, w) gets a new *elementary* connection table C_e only containing elementary connections c_e. Each such connection c_e represents exactly one train T which departs at time $dep_v(time)$ in station $v \in S$ and arrives at arrival time $arr_w(time)$ in station $w \in S$ without stops.

4.2 Arc-Flags

In a time-dependent single-criteria scenario, a set arc-flag $AF_C(a)$ denotes whether e is important for region C. Similar to the augmentations given in [6, 9], we use the following intuition to set an arc-flags in our event-dependent multi-criteria scenario. Set $AF_C = \mathsf{true}$ as soon as e is important for at least one Pareto-path for all possible departure times. In the following, we show how to incorporate this intuition correctly.

Augmentation. A common approach to compute arc-flags in the time-independent single-criteria scenario is based on running DIJKSTRA-queries on the backward graph from each boundary node of the graph. Similarly, we compute event-dependent multi-criteria arc-flags by running our version of DIJKSTRA's algorithm on the backward graph from all departure events of each pre-boundary node b' of boundary node b. Let C be the associated region of b. Note that we run the queries from the pre-boundary nodes. The reason for this is that it simplifies case distinctions considerably. Using boundary nodes instead would require to distinguish between paths ending at the boundary node and paths ending somewhere else within the target region C. Again, like for arc-reduction, we have to use weaker rules of dominance during our queries, given in Table 4 of the Appendix. For all arcs a of the graph, we end up in connection tables representing Pareto paths starting with arc a towards the boundary node b. If the computed connection table of arc a is *not* empty, then a is used for at least one Pareto-path towards C. Hence, we set $AF_C(a)$ to true.

Theorem 3. *Event-dependent multi-criteria arc-flags are correct.*

Unfortunately, classical arc flags turn out to be rather weak: almost all arc flags must be set to true to guarantee correctness of the query algorithm since for any arc there is almost surely one point in time where this arc is part of some Pareto-optimal path towards the target station. However, from our preprocessing we do know exactly at which points of time any particular arc might be necessary. Therefore, we refine the classical arc flags to *time-period arc flags*. The idea is to divide the overall range for which our preprocessing is valid into short time intervals. A good compromise between size of the necessary flags and the desired refinement is to divide a full day into 12 intervals of two hours. Then each arc maintains a flag for each combination of time interval (*period*) and region which tells whether the arc might be "useful" for a particular query within a certain period.

4.3 SHARC

In this work, we use a slightly reduced variant of SHARC. We only use a 1-level setup (due to the limited size of the graphs deriving from our model) and do not use refinement of arc-flags (cf. Section 2). By this, preprocessing is split into three phases. First, we partition the graph into k regions. Then, we perform a route-contraction step according to the above description. Any arc (u, v) bypassed during contraction directly gets its *final* arc-flags assigned, depending on its tail u. If u has been bypassed, (u, v) gets all flags assigned to true, while if u is part of the core, (u, v) gets all flags assigned to false, except for the region v is assigned to, this flag is set to true. Note that in order to guarantee correctness, our route-contraction needs to be region-aware, i.e., a boundary node is never bypassed. After route contraction, we perform an arc-flags preprocessing as stated above on the resulting core. Since we use a setup with one level, our query algorithm is our standard one with a small modification: we only relax arcs which have a time-period arc-flag for the target's region assigned true. However, there is one subtle detail: we have to explore flags for all time periods which can still lead to a Pareto-optimal solution at the target. We use lower bounds on the minimum travel time towards the target to determine which flags we have to consider.

5 Experiments

5.1 Computational Setup

Test data. Our computational study is based on the German train schedule of 2008. This schedule consists of 8817 stations, 40034 trains on 15428 routes, 392 foot paths, and 1,135,479 elementary connections. In our station graph model we obtain a graph with 189,214 arcs. For our tests, we used different types of queries (randomly chosen start stations and destinations, real customer queries, and handmade). The query start interval has been varied between a full day (denoted by [0-24]) and typical two-hour intervals (for example, rush hour [8-10], lunch time [12-14], and late evening [20-22]), as well as one hour [7-8], six hour [6-12], and twelve hour [6-18] intervals.

Environment. All experiments were run on a standard PC (Intel®Core™2 Quad CPU Q6600, 2.4GHz, 4MB cache, 8GB main memory under Ubuntu linux version 9.04. Only one core has been used by our program. Our code is written in C++ and has been compiled with g++ 4.3.3 and compile option -O3.

Preprocessing. Using the graph partitioning library SCOTCH [24] and additional postprocessing by a local optimization routine, we have partitioned the given set of stations into 16 regions. This number of regions seems to be a reasonable compromise between the average region size and the computational effort for the arc flags. The time to compute the partitioning into regions and the time

to compute shortcut arcs is negligible (less than a minute CPU time). The overall arc flag computation, however, is really expensive: it requires 33h 37min but can easily be parallelized. Using all four cores it can be reduced to 8h 40min. We can bypass 5,248 out of 8,817 stations, and 55,742 out of 189,214 original arcs. This leads to the insertion of 19,929 additional shortcut arcs. Flag vectors are quite full, on average 41.4% of their bits are set to 1. This clearly limits the effect which we can expect from arc-flags.

Route vs. station dominance. A crucial point for the efficiency of the query algorithm is the appropriate choice of dominance rules. The stronger the dominance rules, the less priority queue operations have to be performed. However, the application of stronger rules is computationally more expensive. In particular, applying station *and* route dominance turned out to be actually a slow-down in comparison with only using route dominance. Although the combined application of rules saves about 30% of priority queue operations, it almost doubles the computation time. Therefore, we use only route dominance in the following.

Query variants. We compare CPU times and operation counts for the number of priority queue delete-min operations for the following algorithmic variants:

- **base:** the pure multi-dimensional Dijkstra algorithm without any speed-up technique.
- **base+lb:** base plus lower bounds for the domination at the terminal.
- **arc-flags:** base+lb combined with time period arc flags but no shortcuts.
- **greedy arc-flags:** arc-flags with a greedy strategy explained below.
- **SHARC:** arc-flags with shortcuts based on route contraction.
- **SHARC+goal:** SHARC combined with goal direction.
- **greedy SHARC:** SHARC with a greedy strategy explained below.
- **greedy SHARC+goal:** the previous variant combined with goal direction.

The "greedy strategy" does the following: whenever we arrive at some station and consider the next arc, we choose only the very first reachable connection on this arc. In general, this strategy will fail to find all Pareto-optimal paths, but except for somewhat pathological situations we will find for each equivalence class of paths with the same pair of objective values at least one representative.

5.2 Computational Results

Experiment 1: Full day scenario. One primary goal of this project is to provide an efficient range query for a complete day of operation between two arbitrary stations. Table 3 shows the results for this scenario. While our baseline variant base requires an average CPU time of 7.85s, already turning on our lower bound domination reduces the average CPU time to 4.54s. Arc-flags achieve a speed-up of 3.15 over base, and SHARC increases the speed-up further to 4.01 over base. Turning on the greedy strategy yields a speed-up of 7.41 over base for greedy SHARC. The fastest variant is the combination of greedy SHARC with goal-directed search. It reduces the average query time to 0.6s and yields a speed-up factor of 13.08.

Query variant	average CPU time in s	average # pq-min operations	speed-up factor over base CPU time	speed-up factor over base pq-min operations
base	7.85	233,203	1.00	1.00
base+lb	4.54	144,325	1.73	1.62
arc-flags	2.49	130,569	3.15	1.79
SHARC	1.96	95,685	4.01	3.91
SHARC+goal	1.00	52,663	7.85	4.43
greedy arc-flags	1.38	84,444	5.69	2.76
greedy SHARC	1.06	59,589	7.41	3.91
greedy SHARC+goal	0.60	37,867	13.08	6.16

Table 3. Experimental results for a complete day, i.e., the start range interval [0-24].

Experiment 2: Two-hour range queries. In our next experiment we are interested in range queries for two-hour periods in the "morning rush hour" [8-10], at "lunch time" [12-14], and in the "late evening" [20-22]. Detailed results are given in the Appendix, see Tables 5-7. As expected, two-hour range queries are faster than full day queries. While queries for the "morning rush hour" [8-10] and for "lunch time" [12-14] behave very similar — the fastest variant requires 0.27s and 0.29s on average, the "late evening" period is much easier and yields average computation times of 0.13s for greedy SHARC+goal.

Experiment 3: Variation of the range width. We compare the speed-up for different widths of the start interval: 1h, 2h, 6h, 12h, and 24h. Figure 5 shows that the speed-up factors increase with the width of the interval, i.e., the larger the search space the better is the speed-up.

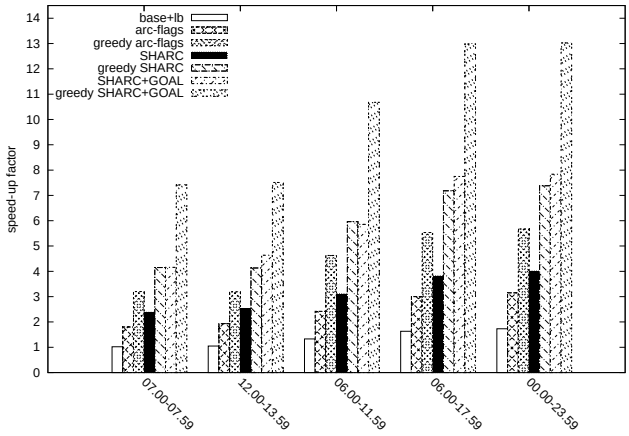

Fig. 5. The speed-up increases with the width of the query interval.

Number of Pareto-optimal paths. For a query range interval of 24h (full day range) we obtain about 7 Pareto-optimal paths on average. Figure 6 shows a histogram for the size of Pareto-optimal paths for the time period of a full day. The maximum number of Pareto-optimal paths which we observe in these tests is 81. An interesting question is whether versions using the greedy strategy or versions using shortcut edges lose any Pareto optima. The good news is that in both cases we have always found the identical set of equivalence classes of Pareto-optimal paths with the same objective values. Differences occur, however, in the total number of alternatives which are identified by these methods. For a full day range, the number of alternatives drops by about 1%. For shorter time periods, the difference is somewhat larger, about 5%.

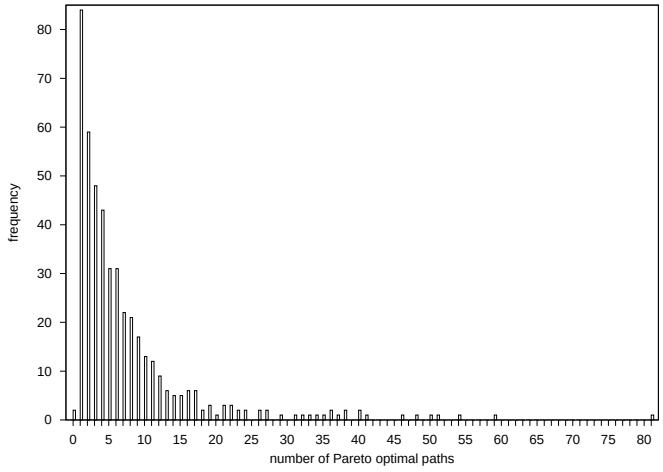

Fig. 6. Frequency of Pareto-optimal paths for a full day range.

6 Conclusion

We presented the first study on advanced speed-up techniques like arc-flags and contraction in a multi-criteria time- and event-dependent scenario which allow us to answer arbitrary range queries. An important lesson we learned from this project is that the classical extension of arc-flags and contraction does not work well. However, with two new concepts, time-period arc flags and route contraction, we can achieve speed-ups of about 13 over the baseline variant for a full day.

It remains an open challenge to develop more powerful speed-up techniques for a multi-criteria time-dependent scenario without scarifying exactness. Since preprocessing for arc flags is very time-consuming, there is also need for tech-

niques which can also be applied in an online scenario where dynamic changes of the schedule are taken into account.

References

1. *Proceedings of ATMOS Workshop 2003*, 2004.
2. R. Bauer and D. Delling. SHARC: Fast and Robust Unidirectional Routing. In I. Munro and D. Wagner, editors, *Proceedings of the 10th Workshop on Algorithm Engineering and Experiments (ALENEX'08)*, pages 13–26. SIAM, April 2008.
3. R. Bauer and D. Delling. SHARC: Fast and robust unidirectional routing. *ACM Journal of Experimental Algorithmics*, 14:2.4–2.29, May 2009. Special Section on Selected Papers from ALENEX 2008.
4. A. Berger and M. Müller-Hannemann. Subpath-optimality of multi-criteria shortest paths in time-dependent and event-dependent networks. Technical report, Martin-Luther-Universität Halle-Wittenberg, Department of Computer Science, 2009.
5. G. Brodal and R. Jacob. Time-dependent Networks as Models to Achieve Fast Exact Time-table Queries. In ATMOS'03 [1], pages 3–15.
6. 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.
7. D. Delling. Time-Dependent SHARC-Routing. *Algorithmica*, July 2009. Special Issue: European Symposium on Algorithms 2008.
8. D. Delling, P. Sanders, D. Schultes, and D. Wagner. Engineering Route Planning Algorithms. In J. Lerner, D. Wagner, and K. A. Zweig, editors, *Algorithmics of Large and Complex Networks*, volume 5515 of *Lecture Notes in Computer Science*, pages 117–139. Springer, 2009.
9. D. Delling and D. Wagner. Pareto Paths with SHARC. In J. Vahrenhold, editor, *Proceedings of the 8th International Symposium on Experimental Algorithms (SEA'09)*, volume 5526 of *Lecture Notes in Computer Science*, pages 125–136. Springer, June 2009.
10. D. Delling and D. Wagner. Time-Dependent Route Planning. In R. K. Ahuja, R. H. Möhring, and C. Zaroliagis, editors, *Robust and Online Large-Scale Optimization*, Lecture Notes in Computer Science. Springer, 2009. Accepted for publication, to appear.
11. Y. Disser, M. Müller–Hannemann, and M. Schnee. Multi-Criteria Shortest Paths in Time-Dependent Train Networks. In McGeoch [19], pages 347–361.
12. R. Geisberger, P. Sanders, D. Schultes, and D. Delling. Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. In McGeoch [19], pages 319–333.
13. F. Geraets, L. G. Kroon, A. Schöbel, D. Wagner, and C. Zaroliagis. *Algorithmic Methods for Railway Optimization*, volume 4359 of *Lecture Notes in Computer Science*. Springer, 2007.
14. T. Gunkel, M. Müller–Hannemann, and M. Schnee. Improved Search for Night Train Connections. In C. Liebchen, R. K. Ahuja, and J. A. Mesa, editors, *Proceedings of the 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'07)*, pages 243–258. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2007.

15. P. Hansen. Bricriteria Path Problems. In G. Fandel and T. Gal, editors, *Multiple Criteria Decision Making – Theory and Application –*, pages 109–127. Springer, 1979.
16. E. Köhler, R. H. Möhring, and H. Schilling. Acceleration of Shortest Path and Constrained Shortest Path Computation. In *Proceedings of the 4th Workshop on Experimental Algorithms (WEA'05)*, Lecture Notes in Computer Science, pages 126–138. Springer, 2005.
17. U. Lauther. An Extremely Fast, Exact Algorithm for Finding Shortest Paths in Static Networks with Geographical Background. In *Geoinformation und Mobilität - von der Forschung zur praktischen Anwendung*, volume 22, pages 219–230. IfGI prints, 2004.
18. E. Q. Martins. On a Multicriteria Shortest Path Problem. *European Journal of Operational Research*, 26(3):236–245, 1984.
19. C. C. McGeoch, editor. *Proceedings of the 7th Workshop on Experimental Algorithms (WEA'08)*, volume 5038 of *Lecture Notes in Computer Science*. Springer, June 2008.
20. R. H. Möhring. Verteilte Verbindungssuche im öffentlichen Personenverkehr – Graphentheoretische Modelle und Algorithmen. In P. Horster, editor, *Angewandte Mathematik insbesondere Informatik, Beispiele erfolgreicher Wege zwischen Mathematik und Informatik*, pages 192–220. Vieweg, 1999.
21. M. Müller–Hannemann and M. Schnee. Finding All Attractive Train Connections by Multi-Criteria Pareto Search. In *Algorithmic Methods for Railway Optimization* [13], pages 246–263.
22. M. Müller–Hannemann, F. Schulz, D. Wagner, and C. Zaroliagis. Timetable Information: Models and Algorithms. In *Algorithmic Methods for Railway Optimization* [13], pages 67–90.
23. M. Müller–Hannemann and K. Weihe. Pareto Shortest Paths is Often Feasible in Practice. In *Proceedings of the 5th International Workshop on Algorithm Engineering (WAE'01)*, volume 2141 of *Lecture Notes in Computer Science*, pages 185–197. Springer, 2001.
24. F. Pellegrini. SCOTCH: Static Mapping, Graph, Mesh and Hypergraph Partitioning, and Parallel and Sequential Sparse Matrix Ordering Package, 2007.
25. E. Pyrga, F. Schulz, D. Wagner, and C. Zaroliagis. Experimental Comparison of Shortest Path Approaches for Timetable Information. In *Proceedings of the 6th Workshop on Algorithm Engineering and Experiments (ALENEX'04)*, pages 88–99. SIAM, 2004.
26. E. Pyrga, F. Schulz, D. Wagner, and C. Zaroliagis. Towards Realistic Modeling of Time-Table Information through the Time-Dependent Approach. In ATMOS'03 [1], pages 85–103.
27. 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.
28. P. Sanders and D. Schultes. Engineering Highway Hierarchies. In *Proceedings of the 14th Annual European Symposium on Algorithms (ESA'06)*, volume 4168 of *Lecture Notes in Computer Science*, pages 804–816. Springer, 2006.
29. F. Schulz. *Timetable Information and Shortest Paths.* PhD thesis, Universität Karlsruhe (TH), Fakultät für Informatik, 2005.
30. D. Theune. *Robuste und effiziente Methoden zur Lösung von Wegproblemen.* PhD thesis, 1995.
31. A. Warburton. Approximation of Pareto Optima in Multiple-Objective Shortest-Path Problems. *Operations Research*, 35(1):70–79, 1987.

Appendix

A Proof of Theorem 2

Theorem 2. *Arc-Reduction preserves event-dependent Pareto-optimal paths between core-nodes.*

Proof. We only prove the correctness for Line 2 of table 2. The other cases can be shown very similarly. We consider two connections c_1 and c_2 on short cut arcs (u, v) which fulfill the conditions in line 2 and column 1. Let P_2 be an event-dependent s,t-path starting at s with earliest start time $start_s$ and ends in t with an artificial departure event end_t at t. Furthermore P_2 contains connection c_2. Let P_{start_s,arr_u} be the event-dependent (s,u)-subpath from P_2 and P_{dep_v,end_t} be the event-dependent (u,t)-subpath from P_2. We denote with $arr_u(route)$ the route number of the arrival event at u and with $dep_v(route)$ the route number of departure event dep_v. We distinguish between four cases.

1. $s = u$ and $v = t$. Then P_{start_s,arr_u} and P_{dep_u,end_t} are empty paths. We construct the event-dependent path P_1 which starts with the earliest start time $start_s$ and ends with the artificial departure event end_t. This is possible because $dep_{time}(c_1) > dep_{time}(c_2)$ is valid. P_1 and P_2 are comparable event-dependent paths and with the conditions in column 2 it follows that $ttime(P_1) < ttime(P_2)$ and $transfer(P_1) < transfer(P_2)$. This implies $P_1 <_{dom} P_2$.
2. $s \neq u$ and $s \neq t$. We distinguish between four different cases.
   (a) $arr_u(route) \neq dep_{c_2}(route)$ and $arr_{c_2}(route) \neq dep_v(route)$. We construct the event-dependent path P_1 which consists of P_{start_s,arr_u}, connection c_1 and P_{dep_u,end_t}. This is possible because it is fulfilled $dep_{time}(c_1) > dep_{time}(c_2)$ and $arr_{time}(c_1) = arr_{time}(c_2)$. P_1 and P_2 are comparable event-dependent paths and with the conditions in column 2 it follows that $ttime(P_1) \leq ttime(P_2)$ and $transfer(P_1) < transfer(P_2)$. This implies $P_1 <_{dom} P_2$.
   (b) $arr_u(route) = dep_{c_2}(route)$ and $arr_{c_2}(route) \neq dep_u(route)$. We construct the event-dependent path P_1 which consists of the maximum event-dependent s,s'-subpath of P_{start_s,arr_u} using routes which are not identical with route dep_{c_2}, then takes the event-dependent s',u-path which uses route $dep_{c_2}(route)$ without transfers and contains connection c_1 and P_{dep_u,end_t}. This is possible because it is fulfilled $dep_{time}(c_1) > dep_{time}(c_2)$, $arr_{time}(c_1) = arr_{time}(c_2)$ and $dep_{c_2}(route) = dep_{c_1}(route)$. This implies at s' a later departure time for P_1. P_1 and P_2 are comparable event-dependent paths and with the conditions in column 2 it follows that $ttime(P_1) = ttime(P_2)$ and $transfer(P_1) < transfer(P_2)$. This implies $P_1 <_{dom} P_2$.
   (c) $arr_u(route) \neq dep_{c_2}(route)$ and $arr_{c_2}(route) = dep_u(route)$. Analogously to case b).

(d) $arr_u(route) = dep_{c_2}(route)$ and $arr_{c_2}(route) = dep_u(route)$. Analogously to case b).

P_1 and P_2 are comparable event-dependent paths and with the conditions in column 2 it follows that $ttime(P_1) = ttime(P_2)$. and $transfer(P_1) < transfer(P_2)$. This implies $P_1 <_{dom} P_2$.

3: $s \neq u$ and $v = t$. Analogously to case 2.

4: $s = u$ and $v \neq t$. Analogously to case 2.

In all four cases we can construct an event-dependent path P_1 which is comparable with P_2, dominates P_2 and does not contain connection c_2. It follows that we can delete connection c_2.

A.1 Dominance Rules for Arc Flag Preprocessing

Table 4 presents the dominance rules which have to be used in the preprocessing phase. Let c_1, c_2 be two connections starting at station v and each ending in a departure event at pre-boundary vertex w.

	c_1, c_2 comparable if	delete $c_2 \Leftrightarrow$
1	$c_1(dep_v(route)) = c_2(dep_v(route))$ $c_1(dep_w(route)) = c_2(dep_w(route))$ $c_1(dep_w(time)) \leq c_2(dep_w(time))$	$ttime(c_1) \leq ttime(c_2)$ $transfer(c_1) < transfer(c_2)$
2	$c_1(dep_w(time)) \leq c_2(dep_w(time))$ $c_1(dep_v(route)) = c_2(dep_v(route))$	$ttime(c_1) \leq ttime(c_2)$ $transfer(c_1) + < transfer(c_2)$
3	$c_1(dep_w(route)) = c_2(dep_w(route))$ $c_1(dep_w(time)) \leq c_2(dep_w(time))$	$ttime(c_1) \leq ttime(c_2)$ $transfer(c_1) + 1 < transfer(c_2)$
4	$c_1(dep_w(time)) \leq c_2(dep_w(time))$	$ttime(c_1) \leq ttime(c_2)$ $transfers(c_1) + 2 < transfers(c_2)$

Table 4. Dominance rules for the preprocessing phase.

B Additional Computational Results

Tables 5-7 show the results of our Experiment 2.

Query variant	average CPU time in s	average # pq-min operations	speed-up factor over base CPU time	speed-up factor over base pq-min operations
base	2.26	71,422	1.00	1.00
base+lb	2.15	68,263	1.05	1.05
arc-flags	1.20	62,291	1.88	1.15
SHARC	0.92	44,060	2.46	1.62
SHARC+goal	0.49	21,645	4.61	3.30
greedy arc-flags	0.65	38,696	3.48	1.85
greedy SHARC	0.51	26,704	4.43	2.67
greedy SHARC+goal	0.27	12,646	8.37	5.65

Table 5. Experimental results for the start range interval [08-10] ("morning rush hour").

Query variant	average CPU time in s	average # pq-min operations	speed-up factor over base CPU time	speed-up factor over base pq-min operations
base	2.17	67,517	1.00	1.00
base+lb	2.07	64,534	1.05	1.04
arc-flags	1.13	57,931	1.92	1.17
SHARC	0.86	40,692	2.52	1.66
SHARC+goal	0.47	20,549	4.62	3.29
greedy arc-flags	0.68	39,052	3.19	1.73
greedy SHARC	0.53	26,905	4.09	2.51
greedy SHARC+goal	0.29	13,583	7.48	4.97

Table 6. Experimental results for the start range interval [12-14] ("lunch time").

Query variant	average CPU time in s	average # pq-min operations	speed-up factor over base CPU time	speed-up factor over base pq-min operations
base	0.41	15,915	1.00	1.00
base+lb	0.39	15,098	1.05	1.05
arc-flags	0.24	14,058	1.71	1.13
SHARC	0.19	9,823	2.16	1.62
SHARC+goal	0.16	7,192	2.56	2.21
greedy arc-flags	0.19	10,893	2.17	1.46
greedy SHARC	0.15	7,586	2.73	2.10
greedy SHARC+goal	0.13	5,472	3.15	2.91

Table 7. Experimental results for the start range interval [20-22] ("late evening").