Fully Dynamic Speed-Up Techniques for Multi-criteria Shortest Path Searches in Time-Dependent Networks*

Annabell Berger, Martin Grimmer, and Matthias Müller-Hannemann

Department of Computer Science, Martin-Luther-University Halle-Wittenberg,
Von-Seckendorff-Platz 1, 06120 Halle, Germany
{berger,muellerh}@informatik.uni-halle.de,
martin.grimmer@particle-dev.com

Abstract. We introduce two new speed-up techniques for time-dependent point-to-point shortest path problems with fully-dynamic updates in a multi-criteria setting. Our first technique, called SUBITO, is based on a specific substructure property of time-dependent paths which can be lower bounded by their minimal possible travel time. It requires no preprocessing, and the bounds can be computed on-the-fly for each query. We also introduce k-flags, an extension of arc flags, which assigns to each arc one of k levels for each region of a vertex partition. Intuitively, the higher the level of an arc for a certain destination, the larger the detour with respect to travel time. k-flags allow us to handle dynamic changes without additional time-consuming preprocessing.

In an extensive computational study using the train network of Germany we analyze these and other speed-up techniques with respect to their robustness under high and realistic update rates. We show that speed-up factors are conserved under different scenarios, namely a typical day of operation, distributed delays after "heavy snowfall", and a major disruption at a single station. In our experiments, k-flags combined with SUBITO have led to the largest speed-up factors, but only marginally better than SUBITO alone. These observations can be explained by studying the distribution of k-flags. It turns out that only a small fraction of arcs can be excluded if one wants to guarantee exact Pareto-optimal point-to-point queries.

Keywords: Shortest paths; dynamic graphs; speed-up technique.

1 Introduction

Up to now, route planning is usually based on a static road map or a published schedule in public transport. Unfortunately, car traffic is affected by time-dependent speed profiles (like every day rush hours) and unforeseeable events,

* 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.

P. Festa (Ed.): SEA 2010, LNCS 6049, pp. 35–46, 2010.

like traffic jams, and blocked roads because of construction work. Likewise, public transportation systems suffer from delays for various reasons. While speed profiles can be modelled quite well by time-dependent models, unforeseeable events like traffic jams and train delays require in addition a truly dynamic model to support on-trip routing.

The demands for an online tool are quite high: To give a concrete example, on a typical day of operation of German Railways, an online system has to handle about 6 million forecast messages about (mostly small) changes with respect to the planned schedule and the latest prediction of the current situation. Initial primary train delays induce a whole cascade of secondary delays of other trains which have to wait according to certain waiting policies between connecting trains. Thus, the graph model has to be updated at a rate of about 70 update operations per second [1]. From the traveler's point of view, a navigation or timetable information system should offer a choice of reasonable alternatives subject to optimization criteria like earliest arrival time, travel costs, sustainability, or traveler comfort. Hence, the task is to solve a multi-criteria optimization problem, i.e., to find all Pareto-optimal itineraries. Since central timetable information servers have to answer millions of queries per day, it is obvious that low query times are crucial. The algorithmic challenge is to design speed-up techniques which are provably correct and work in a multi-criteria fully-dynamic scenario.

In recent years, a multitude of powerful speed-up techniques have been developed for point-to-point shortest path queries in static transportation networks. These techniques work empirically extremely well for road networks, yielding query times of a few microseconds in continental-sized networks, for a recent survey see [2]. Quite recently, Abraham et al. [3] complemented these observations with the first rigorous proofs of efficiency for many of the speed-up techniques suggested over the past decade. They introduced the notion of highway dimension and showed that networks with small highway dimension have good query performance. In contrast, several empirical studies have shown that these speed-up techniques work less well for public transportation networks (with scheduled traffic for trains, buses, ferries, and airplanes) [4,5]. To understand this difference, it is helpful to consider time-expanded graph models where each arrival or departure event corresponds to a vertex. These graphs are highly non-planar and have relatively high highway dimension. A detailed explanation of further reasons why timetable information in public transport is more challenging than routing in road networks has been given by Bast [6].

Speed-up techniques in a dynamic context. The general challenge is to design a speed-up technique based on preprocessing with the following features:

1. Even after many dynamic graph changes, we do not require a further time-consuming preprocessing. Moreover, the speed-ups in a time-dependent scenario with and without dynamic updates should be comparable.
2. The method should allow us to find *all* Pareto-optimal paths within a reasonable range, not only one representative for each Pareto-optimal solution vector.

Related Work. The literature on point-to-point shortest path problems uses the term dynamic in several different ways, leading to problem variants which can be categorized as follows:

Problem version 1: ("static shortest path problem with cost updates") This model considers a static graph with cost updates. Static here refers to the arc costs which are constant between two consecutive updates. This version can be seen as a first step towards modelling traffic jams or blocked roads in road networks. However, the shortcoming of this model is that it does not capture the time dimension correctly. For example, if some bridge or road is blocked for half an hour, it may or may not have an influence for a routing request, depending on the point of time when one could reach this place. So at query time, the static graph includes only one of these two possibilities.

Problem version 2: ("time-dependent earliest arrival problem with dynamic updates") This model is built on a time-dependent graph, where each arc has a discrete travel-time function. The travel functions can be used to model speed profiles. These functions are dynamically updated at discrete time points. Updates are unplanned changes of the speed profile, for example due to a blocked bridge as above. This version is usually treated as a single-criterion optimization problem with the objective to find a route with earliest arrival. A standard approach is to look only for *greedy paths*. In road networks, this means that the traveler immediately decides at a junction which road to take next (he does not wait for better alternatives). In public transportation, greedy means that the passenger takes the very first possibility on each line. Unfortunately, optimal greedy paths can be sub-optimal in the set of all time-dependent paths in a dynamic scenario. In road networks, it can now be reasonable to wait before a blocked bridge for some minutes to get the faster path. Thus the problem is closer to public transport where waiting times have to be considered already in an undelayed scenario. These complications can be handled, but such queries require a larger search effort for non-greedy paths [7].

Problem version 3: ("fully-dynamic multi-criteria on-trip problem") This version comprises discrete event-dependent multi-criteria search for all shortest (s, t)-paths where one optimization criterion corresponds to the earliest arrival time. If an additional optimization criterion depends on other attributes than time, the problem becomes event-dependent (for example, on the specific train). Fully dynamic here means that the travel time functions can be varied freely, including arc deletions or temporarily blocked arcs.

Work on problem version 1 includes, for example, dynamic shortest path containers [8], dynamic highway node routing [9], landmark-based routing in dynamic graphs [10], dynamic arc flags [11], and contraction hierarchies [12].

Problem version 2 has found much less attention. Nannicini et al. [13] have chosen a slightly different approach to handle a realistic dynamic situation in road networks where traffic information data are updated each minute. They relax the problem of finding shortest paths to determining paths which are very close to an optimal solution. Their underlying graph model can be seen as

time-dependent, the travel time functions are determined heuristically based on real-time traffic data. Unfortunately, they cannot give a guarantee for the optimality of a solution. Surprisingly, the proposed solution works well in practice, with 0.55% average deviation from the optimal solution and a recorded maximum deviation of 17.59%. In another paper, Nannicini et al. [14] proposed a two-phase polynomial time approximation scheme for the point to point shortest path problem. In summary, there is no exact method available which has been tested for problem version 2, and to the best of our knowledge, problem version 3 is first studied in this paper.

Our contribution. Our approach makes use of the following assumption which is valid in a railroad scenario: the travel time on each time-dependent arc can be lower-bounded by some positive value which we call the minimum-theoretical travel time. Above this lower bound value, the travel time function can be freely modified by dynamically changing delays. In particular, we allow the symbolic value $+\infty$ to model the unavailability of an arc. We only forbid dynamic updates which insert new arcs into our model. This is no strong restriction since new railroad tracks will not be provided all of a sudden, they will usually be known right in advance to be included into the preprocessing. Lower travel time bounds on arcs extend to lower bounds on time- and event-dependent paths. This leads us to the new notion of additive c-optimality for static and event-dependent paths. Its properties are useful to restrict the search space with respect to a concrete (s, t)-query. We introduce two new speed-up techniques which we call *SUBITO* and *k-flags*. The speed-up technique SUBITO reduces the number of events in a timetable and can be used on-the-fly without time-consuming preprocessing. k-flags allow us to restrict the search to a subgraph after a preprocessing phase. Both techniques can also be combined. For the first time, these techniques meet all mentioned goals for a dynamic context.

The proposed techniques have been experimentally evaluated on the German train networks with several realistic delay scenarios (a typical day of operation, distributed delays due to bad weather like a "snow chaos", and a major disruption at a single station). We observe a strong positive correlation of the observed running time and the travel time. The best speed-ups are obtained for the most difficult queries, i.e., those with large travel times. We also show that speed-up factors are conserved under high update rates. In our experiments k-flags combined with SUBITO have led to the largest speed-up factors, but only marginally better than SUBITO alone. These observations can be explained by studying the distribution of k-flags. It turns out that only a small fraction of arcs can be excluded if one wants to guarantee exact Pareto-optimal point-to-point queries.

Overview. The remainder of the paper is structured as follows. In Section 2, we discuss the on-trip problem and briefly review our concept of time- and event-dependent paths and provide basic definitions. Our main contribution — the speed-up techniques SUBITO and k-flags — are introduced in Section 3. In Section 4, we give the results of our experiments testing our speed up techniques with real-time traffic data of German Railways.

2 The On-Trip Problem

2.1 Graph Models

A timetable $TT := (Z, S, C)$ consists of a tuple of sets. Let Z be the set of trains, S the set of stations and C the set of *elementary connections*, that is

$$C := \left\{ c = (z, s, s', t_d, t_a) \,\middle|\, \begin{array}{l} \text{Train } z \in Z \text{ leaves station } s \text{ at time } t_d. \\ \text{The next stop of } z \text{ is at station } s' \text{ at time } t_a. \end{array} \right\}.$$

A timetable TT is valid for a number of N traffic days. A timetable-valid function $v : Z \to \{0,1\}^N$ determines on which traffic days the train operates. We denote a time value t in TT by $t := a \cdot 1440 + b$ with $a \in [0, N], b \in [0, 1439]$. The actual time within a day is then $t_{day} = t \ mod \ 1440$ and the actual day $d = \lfloor \frac{t}{1440} \rfloor$. We define the *station graph* $G = (V, A)$ with $V := S$ and $A := \{(v,u) |\ v, u \in V,\ \exists c \in C(TT) \text{ from } v \text{ to } u\}$. For several reasons it is useful to construct an extended version of the station graph by splitting each arc of the station graph G into parallel arcs. We denote a sequence $(v_1, \ldots, v_l)$ of pairwise disjoint vertices $v_i \in V$ as a *route* r if there exists at least one train using exactly these stations v_i in the given order. This setting allows us to construct the so-called *station route graph* $G_r = (V, A_r)$ with $A_r := \{(v,u)_r \mid (v,u) \in A, r \text{ is route in TT using arc } (v,u)\}$.

Let $T \subset \mathbb{N}$ be a set of time points. With respect to the set of elementary connections C we define a set of departure events Dep_v and arrival events Arr_v for all $v \in V$. Each event $dep_v := (time, train, route) \in Dep_v$ and $arr_v := (time, train, route)$ represents exactly one departure or arrival event which consists of the three attributes *time*, *train* and *route*. We assign to each arc $a = (v,u) \in A_r$ and departure event dep_v at v an arrival event arr_u which defines the event that we reach vertex u if we depart in v with departure event dep_v and traverse arc a on route r. This setting represents a corresponding elementary connection $c \in C$. Hence, the event dep_v departs at time $dep_v(time) \in \mathbb{N}$ in v, travels on the arc $a = (v,u) \in A$ and arrives at the corresponding event arr_u at $arr_u(time) \in T$. Staying in any vertex can be limited to minimum and maximum staying times $minstay(arr_v, dep_v), maxstay(arr_v, dep_v) \in \mathbb{R}_+$ which have to be respected between different events in v. Staying times ensure the possibility to transfer from one train to the next. The following definitions are given with respect to an (s,t)-query for an earliest start time $start_s(time)$ at s. For an (s,t)-query we ignore all arrival events at s, but introduce an artificial "arrival event" $start_s$ with an earliest start time $start_s(time) \in \mathbb{N}$ which can be interpreted as an arrival time. Furthermore, we define one artificial "departure event" end_t which can be interpreted as a departure time in t.

Event-dependent vertices v_e are 2-tuples which consist of an arrival event $arr_v \in Arr_v$ and a departure event $dep_v \in Dep_v$ for a vertex $v \in V$, respecting minimum and maximum staying times $minstay, maxstay$ at v. *Event-dependent edges* a_e are tuples which consist of a departure event at $v \in V$ and the corresponding arrival event at $u \in V$ by traversing the edge $(v,u) \in A_r$. These notions have been introduced in [7]. We define a weighted discrete event-dependent graph

$\mathcal{G}(G_r, s, t, start_s) := (V_E, A_E, w)$ with respect to an (s,t)-query with an earliest start time $start_s(time)$ and underlying station route graph G_r. It is a tuple consisting of the set of event-dependent vertices V_E, the set of event-dependent edges A_E and a weight-function $w \in (\mathbb{R}^k)^{V_E \cup A_E}$. The weight-function w assigns to each event-dependent vertex v_E a k-dimensional vector $w(v_e) \in \mathbb{R}^k$ and to each event-dependent edge a_e a k-dimensional vector $w(a_e) \in \mathbb{R}^k$.

2.2 On-Trip Paths

Definition 1. *An* event-dependent path P_E *is an alternating sequence of event-dependent vertices and edges with event-dependent vertices on its ends. The stations of the event-dependent vertices in P_E are pairwise different. We call two event-dependent paths* comparable *if their first event-dependent vertices possess an identical arrival event and their last event-dependent vertices possess an identical departure event.*

We distinguish between two notations for event-dependent paths. If we focus on the origin v and the destination u of a path ("stations"), we call it *event-dependent (v,u)-path*. On the other hand if we focus on starting and ending events of a path we denote an event-dependent path which starts with the event-dependent vertex $v_T = (arr_v, dep_v)$ and ends with the event-dependent vertex $u_T = (arr_u, dep_u)$ as P_{arr_v, dep_u}. Note that a single event-dependent vertex $v_T = (arr_v, dep_v)$ is already an event-dependent path P_{arr_v, dep_v}. All event-dependent (s,t)-paths in our definition are comparable because they possess identical arrival events $arr_s := start_s$ and identical departure-events $dep_t := end_t$. Following Orda and Rom [15] we define a function *top* which assigns to each event-dependent path P_E its underlying sequence p of vertices $v \in V$. We call p *topological path of* P_E and denote it by $top(P_E) =: p$. Obviously, p is a simple path in the station-route-graph G_r, because P_E is vertex-disjoint due to Definition 1. For railway networks, we use the following weight-function $ontrip := (eatt, transfers) : V_E \cup A_E \to \mathbb{R}_+$ for all event-dependent vertices $v_e := (arr_v, dep_v) \in V_E$ and for all time-dependent arcs $a_e := (dep_v, arr_u) \in A_E$.

$$eatt(v_e) := \begin{cases} dep_v(time) - arr_v(time), & \text{for all } v \in V \setminus \{s,t\} \\ dep_s(time) - start_s(time), & \text{for } v = s \\ 0, & \text{for } v = t. \end{cases}$$

$$eatt(a_e) := \Big\{ arr_u(time) - dep_v(time), \quad \text{for all } v \in V.$$

$$transfers(v_e) := \begin{cases} 1 & \text{for all } v \in V \setminus \{s,t\} \text{ with } dep_v(train) \neq arr_v(train) \\ 0 & \text{otherwise.} \end{cases}$$

$$transfers(a_e) := 0$$

The first function *eatt* computes the *earliest arrival travel time*, it includes travel times on arcs and staying times at stations. The second function $transfers$ counts the number of transfers between trains. For an event-dependent path

P_E we define $ontrip(P_E) := \sum_{v_e \in V_E} ontrip(v_e) + \sum_{a_e \in A_E} ontrip(a_e)$. Pareto-optimal on-trip paths can be computed by a Dijkstra-like labeling algorithm, see [7] for details. We would like to point out that Pareto-optimal on-trip greedy paths are not necessarily Pareto-optimal on-trip paths in the station graph. Fortunately, one can show that the set of Pareto-optimal on-trip greedy paths is a subset of all Pareto-optimal on-trip paths in the station route graph G_r.

3 Speed-Up Techniques SUBITO and *k*-Flags

3.1 *c*-Optimality and the SUBITO Technique

In this section we consider a dynamic timetable, i.e., one with dynamic changes for departure and arrival events. In particular, departure times and arrival times can change. Let $G_r = (V, A_r)$ be the station route graph with a static weight g. Let $V^* := \{V^1, \ldots, V^\nu\}$ be a partition of vertex set V, each element $V^j \in V^*$ is called a *region*. Furthermore, we denote all vertices $b \in V^j$ as *boundary vertices of region* V^j if and only if there exists at least one arc $(b, u), (u, b) \in A_r$ with $u \notin V^j$. We define the arc-weight-function $g : A_r \to \mathbb{R}^+$, denoting the *minimal theoretical travel time* on arc $a \in A_r$ on the station route graph G_r. Furthermore, we denote the minimal theoretical travel time on a topological path p by $g(p) := \sum_{a \in p} g(a)$. We denote by $d_g(s,t) := \min_{p \in p_{st}} g(p)$ the *smallest* minimal theoretical travel time between two stations s and t. We call a topological (s,t)-path p with $d_g(s,t) \leq g(p) \leq d_g(s,t) + c$ a *c-optimal topological path.* These definitions give us the desired substructure properties of c-optimality.

Theorem 1 (substructure property of *c*-optimal paths). *Let p' be a topological subpath of an (s,t)-topological path p in the station route graph $G_r = (V, A_r, g)$. Then we have: p is c-optimal $\Rightarrow p'$ is c-optimal.*

Next we define a subset of event-dependent paths. We call an event-dependent path P_{arr_u, dep_v} with $eatt(P_{arr_u, dep_v}) \leq d_g(v,v) + c$ and $c \in \mathbb{R}_+$ an *event-dependent c-optimal* path. This definition relates static and event-dependent paths. Similar to static c-optimal paths we also observe a substructure property for c-optimal event-dependent paths.

Theorem 2 (event-dependent *c*-optimality and substructure property) *Let P_{start_s, end_t} be an event-dependent c-optimal path in $\mathcal{G}(G_r, s, t, start_s)$ with $eatt(P_{start_s, end_t}) \leq d_g(s,t) + c$. Then each subpath P_{start_s, dep_u} is an event-dependent c-optimal path with $eatt(P_{start_s, dep_u}) \leq d_g(s,u) + c$.*

Our goal is to find all on-trip paths arriving at most max_{time} time after the earliest possibility. In the first step of our algorithm we determine the smallest minimum theoretical travel times $d_g(s,v)$ for all $v \in V$. In a next step we determine the earliest arrival time at t with respect to the earliest start time $start_s(time)$ in s. Then, we determine the maximum required value c_{max} by computing $c_{max} := eatt + max_{time} - d_g(s,t)$. Finally, we apply a Dijkstra-like labeling algorithm where we now can ignore departure events dep_u which do not fulfill $dep_u(time) \leq d_g(s,u) + c_{max}$. We denote these speed-up technique as *SUBITO-technique* for *sub*structure *i*n *t*ime-dependent *o*ptimization.

3.2 The Idea and Correctness of *k*-Flags

In this section we introduce the notion of *k-flags* which generalize classical arc-flags.

Definition 2 (*k*-flags). *Let $G_r = (V, A_r, g)$ be the station route graph, V^* a vertex partition of V, $c_i \in \mathbb{N}_+ \cup \{0\}$ with $c_1 < \cdots < c_k$, and $i \in \{1, \cdots, k\}$. We define for each $a = (v, u) \in A_r$ the k-flag function $f_a \in \{1, \cdots, k\}^{V^*}$* with

$$f_a(V^j) = \min\{i \mid \exists\ t \in V^j \text{ and a } c_i\text{-optimal path } p = (v, u, \cdots, t)\}.$$

Note, that there may exist vertices v with $d_g(v, t) = \infty$ for all $t \in V^j$. In this case all arcs (v, u) do not get a k-flag for region V^j. We construct a sequence of *query graphs* $G^i_{V^j} := (V^i_{V^j}, A^i_{V^j})$, $i \in \{1, \cdots, k\}$ with respect to an (s, t)-query. We define $A^i_{V^j} := \{a \in A_r \mid f_a(V^j) \leq i, t \in V^j\}$ and $V^i_{V^j} := \{v \in V \mid (v, u) \in A^i_{V^j} \ \vee (u, v) \in A^i_{V^j}\}$. It follows $A^i_{V^j} \subset A^{i+1}_{V^j} \subset A_r$ for all $i \in \{1, \cdots, k-1\}$. The next theorem proves that we find all c_i-optimal topological (s, t)-paths p in query graph $G^i_{V^j}$. This reduces the number of arcs in our station route graph for the search of c_i-paths with $i < k$.

Theorem 3 (consistence of c-optimality). *Let $G_r = (V, A_r, g)$ be the station route graph and $G^i_{V^j} := (V^i_{V^j}, A^i_{V^j})$ the query-graph from level i. Then it follows: p is c_i-optimal topological (s, t)-path in $G_r = (V, A_r) \Leftrightarrow$ p is c_i-optimal topological (s, t)-path in $G^i_{V^j} := (V^i_{V^j}, A^i_{V^j})$.*

The next theorem shows that we are able to find all optimal on-trip paths (with an arrival time at most c_i time units after the fastest path) in the event-dependent graph possessing as underlying station graph our reduced query graph.

Theorem 4 (arc reduced event-dependent query graph). *Let $G_r = (V, A_r)$ be the station route graph, $\mathcal{G}(G_r, s, t, start_s)$ the corresponding event-dependent graph. Let $G^i_{V^j} = (V^i_{V^j}, A^i_{V^j})$ the query graph, $\mathcal{G}(G^i_{V^j}, s, t, start_s)$ the corresponding event-dependent graph. Then it follows: P_{arr_v, dep_u} is an event-dependent c_i-optimal path in $\mathcal{G}(G_r, s, t, start_s)$*
$\Rightarrow P_{arr_v, dep_u}$ is an event-dependent c_i-optimal path in $\mathcal{G}(G^i_{V^j}, s, t, start_s)$.

SUBITO and k-flags can be combined, that means, we search for event-dependent c_i-optimal paths in $\mathcal{G}(G^i_{V^j}, s, t, start_s)$ where we ignore all event-dependent (s, u)-subpaths P_{start_s, dep_u} which are not c_i-optimal. The correctness of this approach follows by Theorems 2 and 4. See a pseudocode in Algorithm 1.

4 Experimental Study

Test instances and environment. 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 route graph model we obtain a graph with 189,214 arcs. For our tests, we used queries with randomly chosen start stations and destinations, and different earliest start times (namely 0:00, 4:00, 8:00, 12:00, 16:00, 20:00).

Algorithm 1. On-Trip-Algorithm with SUBITO k-flags

Input: Origin s, destination $t \in V^j$, earliest start event $start_s$, preprocessed schedule, parameter $max_{time} \geq 0$.

Output: Set of c_i-optimal on-trip paths for $i = 1, \ldots, i_{max} \leq k$. *//Let P_{start_s,end_t} be an event-dependent Pareto-optimal path with minimum eatt. We determine all Pareto-optimal on-trip paths P'_{start_s,end_t} with $eatt(P'_{start_s,end_t}) \leq eatt(P_{start_s,end_t}) + max_{time} \leq d_g(s,t) + c_{i_{max}}$.*

1: Compute the minimum theoretical travel time $d_g(s,v)$ for all $v \in V$ in G_r.
2: Compute the minimal earliest arrival travel time (eatt) at $t \in V$. *//Do a single-criteria event-dependent Dijkstra search with respect to eatt level by level until t is settled.*
3: Compute the maximum required level i_{max} with respect to max_{time}.
4: Compute Pareto-optimal on-trip-paths with a label setting algorithm (see [7]) on the query graph $G^{i_{max}}_{V^j}$. Ignore departure events which do not possess $c_{i_{max}}$-optimality (SUBITO).

We considered three types of delay scenarios:

1. We used the complete delay information of a typical day of operation in Germany available at 8:00 a.m.
2. We simulated a number of considerable delays distributed over all Germany. Our experiments are parameterized with the number $x \in \{10, 50, 100, 150\}$ of primary delays and the delay size $s \in \{5, 10, \ldots, 60\}$.
3. We simulated a major disruption at a single station, more precisely, we stopped all trains passing through this station by two hours. This scenario models disruptions due to a power failure, the breakdown of a signal tower, or the evacuation of a station after a bomb threat.

For the simulation of delays we used the prototype MOTIS (multi-objective traffic information system) which provides a fully realistic model using the standard waiting policies of German Railways to calculate secondary delays. MOTIS can handle massive data streams [1]. All experiments were run on a standard PC (Intel(R) Xeon(R), 2.93GHz, 4MB cache, 47GB main memory under ubuntu linux version 8.10). Only one core has been used by our program. Our code is written in C++ and has been compiled with g++ 4.4.1 and compile option -O3.

Algorithmic variants and combinations. We used the following variants:

- **base:** This variant is a reasonably tuned multi-criteria Dijkstra-like algorithm which uses the speed-up technique dominance by results in combination with lower travel time bounds towards the destination (cf. [4]).
- **goal:** This version adds to base a version of goal-directed search.
- **SUBITO** refers to our new speed-up technique.
- **kFlags** refers to a version which combines all other techniques with k-flags.
- If we search only for greedy paths, we denote this by the attribute **greedy**.

Each variant is parameterized by max_{time}, i.e., we search for paths which arrive at most max_{time} minutes after the earliest possibility at the destination.

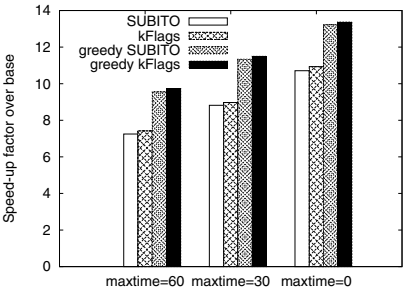

Fig. 1. Comparison of different speed-up techniques

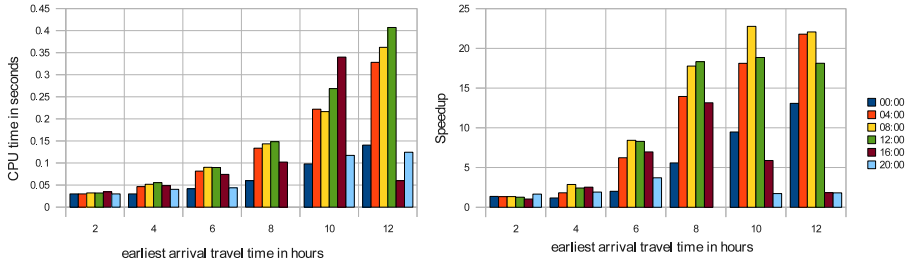

Fig. 2. Experiment 1: Results of greedy k-flags with $max_{time} = 0$. Each column corresponds to a class of queries with the same earliest start time.

Preprocessing. SUBITO either requires no preprocessing at all (the variant used in our experiments) or a single all-pairs shortest path computation on the station route graph which takes less than two minutes (and requires no update after dynamic changes). The computation of k-flags is computationally expensive, it requires about 3h 40 minutes for 128 regions.

Experiment 1: No delays. In our first experiment, we are interested in the comparison of variants in a scenario without delays. In Figure 1, we display the results of the base variant in comparison with goal-direction, SUBITO and kFlags (with and without greedy heuristic), and $max_{time} \in \{0, 30, 60\}$. The baseline variant requires an average CPU time of $1.63s$. Turning on goal-direction, already reduces the average CPU time to $0.25s$. For fixed parameter max_{time}, the kFlags variant is consistently the fastest method, but only marginally faster than SUBITO (an explanation follows below). Searching for greedy paths allows a significant speed-up over non-greedy paths. It is interesting to observe that CPU times and speed-ups depend on the earliest start time and the earliest arrival travel time (eatt). Figure 2 shows this dependency. The longer the travel time, the more CPU time is needed, but speed-ups are also higher for such queries.

Experiment 2 (delays on an ordinary day of operation). We have run an experiment with original delay data for a typical day of operation. Using several thousands of passenger queries, we observed almost no measurable effect on the running time. A corresponding table is therefore omitted.

Experiment 3 (distributed delays "snow chaos"). The purpose of our next experiment is to analyze whether the speed-up techniques are robust against an increasing number of update operations. We varied the number of primary delays from 10 to 150, leading to 188 and 4978 graph update operations, respectively. While several previous computational studies observed a major drop down of speed-ups already after a few update operations [8,11], there is almost no measurable effect on the CPU time and speed-up factors when we increase the number of primary delays.

Experiment 4 (closing down a central station for two hours). Next we analyzed the effect of stopping all trains passing through some central station for two hours (from 11:55 to 13:55). For that purpose, we selected three major German railway stations (Frankfurt am Main Hbf, Hannover Hbf, and Leipzig Hbf). For this experiment, we use a special set of 1000 queries each, designed such that it is likely that the fastest route usually would pass the closed station. Indeed, we observed a posteriori that a high percentage of passengers were affected and had to suffer from an increased earliest arrival time. For example, in the case of Frankfurt, even 98% of the passengers arrived later than without delays. As a consequence, queries require slightly more CPU time, but the speed-up factors over the baseline variant increase even slightly, see Table 1.

Table 1. Closing down a major station (Frankfurt a.M. Hbf, Hannover Hbf, Leipzig Hbf) for two hours. Average query time in seconds and speed-up factors over base.

Variant	Frankfurt a.M. Hbf		Hannover Hbf		Leipzig Hbf	
	CPU	speed-up	CPU	speed-up	CPU	speed-up
base	2.21	1.00	2.69	1.00	2.49	1.00
goal	0.79	2.80	0.34	7.91	0.27	9.27
SUBITO, $max_{time} = 60$	0.24	9.11	0.28	9.78	0.23	10.76
kFlags, $max_{time} = 60$	0.24	9.32	0.27	9.96	0.23	10.92
greedy, kFlags, $max_{time} = 60$	0.19	11.93	0.20	13.54	0.18	14.20
greedy, kFlags, $max_{time} = 0$	0.13	17.39	0.15	17.93	0.13	19.18

Analysis of k-flag distribution. The distribution of k-flags sheds some light on the potential speed-up which can be gained by techniques which try to thin out the station route graph.

For train networks we observe that a multi-criteria search requires a large fraction of arcs in order to guarantee exact solutions for point-to-point queries. The average maximal required c-value c_{max} for our queries is $c_{max} = 143$, which means that the underlying search subgraph for the k-flag technique consists of about 72% of all arcs. About 25% of all arcs can be excluded since the destination cannot be reached on simple paths.

References

1. Müller-Hannemann, M., Schnee, M.: Efficient timetable information in the presence of delays. In: Zaroliagis, C. (ed.) Robust and Online Large-Scale Optimization. LNCS, vol. 5868, pp. 249–272. Springer, Heidelberg (2009)
2. Delling, D., Sanders, P., Schultes, D., Wagner, D.: Engineering route planning algorithms. In: Lerner, J., Wagner, D., Zweig, K.A. (eds.) Algorithmics of Large and Complex Networks. LNCS, vol. 5515, pp. 117–139. Springer, Heidelberg (2009)
3. Abraham, I., Fiat, A., Goldberg, A.V., Werneck, R.F.: Highway dimension, shortest paths, and provably efficient algorithms. In: SODA 2010: Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 782–793. SIAM, Philadelphia (2010)
4. Berger, A., Delling, D., Gebhardt, A., Müller-Hannemann, M.: Accelerating time-dependent multi-criteria timetable information is harder than expected. In: Clausen, J., Stefano, G.D. (eds.) ATMOS 2009 - 9th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Germany (2009)
5. Bauer, R., Delling, D., Wagner, D.: Experimental study on speed-up techniques for timetable information systems. Networks (to appear, 2010)
6. Bast, H.: Car or public transport—two worlds. In: Albers, S., Alt, H., Näher, S. (eds.) Efficient Algorithms: Essays Dedicated to Kurt Mehlhorn on the Occasion of His 60th Birthday. LNCS, vol. 5760, pp. 355–367. Springer, Heidelberg (2009)
7. Berger, A., Müller-Hannemann, M.: 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)
8. Wagner, D., Willhalm, T., Zaroliagis, C.D.: Dynamic shortest paths containers. Electr. Notes Theor. Comput. Sci. 92, 65–84 (2004)
9. Schultes, D., Sanders, P.: Dynamic highway-node routing. In: Demetrescu, C. (ed.) WEA 2007. LNCS, vol. 4525, pp. 66–79. Springer, Heidelberg (2007)
10. Delling, D., Wagner, D.: Landmark-based routing in dynamic graphs. In: Demetrescu, C. (ed.) WEA 2007. LNCS, vol. 4525, pp. 52–65. Springer, Heidelberg (2007)
11. Berrettini, E., D'Angelo, G., Delling, D.: Arc-flags in dynamic graphs. In: Clausen, J., Stefano, G.D. (eds.) ATMOS 2009 - 9th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Germany (2009)
12. Geisberger, R., Sanders, P., Schultes, D., Delling, D.: Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In: McGeoch, C.C. (ed.) WEA 2008. LNCS, vol. 5038, pp. 319–333. Springer, Heidelberg (2008)
13. Nannicini, G., Baptiste, P., Krob, D., Liberti, L.: Fast computation of point-to-point paths on time-dependent road networks. In: Yang, B., Du, D.-Z., Wang, C.A. (eds.) COCOA 2008. LNCS, vol. 5165, pp. 225–234. Springer, Heidelberg (2008)
14. Nannicini, G., Baptiste, P., Barbier, G., Krob, D., Liberti, L.: Fast paths in large-scale dynamic road networks. Computational Optimization and Applications (published online, 2008)
15. Orda, A., Rom, R.: Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. J. ACM 37, 607–625 (1990)