Engineering Graph-Based Models for Dynamic Timetable Information Systems*

Alessio Cionini[1], Gianlorenzo D'Angelo[2], Mattia D'Emidio[1], Daniele Frigioni[1], Kalliopi Giannakopoulou[3,4], Andreas Paraskevopoulos[3,4], and Christos Zaroliagis[3,4]

1 Department of Information Engineering, Computer Science and Mathematics, University of L'Aquila, Italy.
alessio.cionini@gmail.com, {mattia.demidio, daniele.frigioni}@univaq.it
2 Gran Sasso Science Institute (GSSI), L'Aquila, Italy.
gianlorenzo.dangelo@gssi.infn.it
3 Computer Technology Institute and Press "Diophantus", Patras, Greece.
4 Department of Computer Engineering and Informatics, University of Patras, 26504 Patras, Greece. {gianakok,paraskevop,zaro}@ceid.upatras.gr

Abstract

Many efforts have been done in the last years to model public transport timetables in order to find optimal routes. The proposed models can be classified into two types: those representing the timetable as an *array*, and those representing it as a *graph*. The array-based models have been shown to be very effective in terms of query time, while the graph-based models usually answer queries by computing shortest paths, and hence they are suitable to be used in combination with speed-up techniques developed for road networks.

In this paper, we focus on the *dynamic* behavior of graph-based models considering the case where transportation systems are subject to delays with respect to the given timetable. We make three contributions: (i) we give a simplified and optimized update routine for the well-known time-expanded model along with an engineered query algorithm; (ii) we propose a new graph-based model tailored for handling dynamic updates; (iii) we assess the effectiveness of the proposed models and algorithms by an experimental study, which shows that both models require negligible update time and a query time which is comparable to that required by some array-based models.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G2.2.2 Graph Theory, G.4 Mathematical Software

Keywords and phrases Timetabling, dynamic updates, queries, shortest paths

Digital Object Identifier 10.4230/OASIcs.ATMOS.2014.46

1 Introduction

Computing the best route in a public transportation system is a problem faced by everybody who ever traveled. Nowadays, public transportation companies have on-line journey planners which are able to answer to queries like "What is the *best* route from some station A to some other station B if I want to depart at time t?". Usually the best route is the one

* Research partially supported by the Italian Ministry of University and Research under Research Grants 2010N5K7EB PRIN 2010 (ARS TechnoMedia) and 2012C4E3KT PRIN 2012 (AMANDA), as well as by EU FP7/2007-2013 under grant agreements no. 288094 (eCOMPASS), no. 609026 (MOVESMART), and no. 621133 (HoPE).

that minimizes the traveling time (*earliest arrival time problem*), or the number of times that a passenger has to move from one train to another one (*minimum number of transfers problem*), or both the previous objective function (*multi-criteria problem*). The input of such problems is given by a *timetable* which consists of a set of stations (e.g. train stations, bus stops, etc.), a set of vehicles (trains, buses, etc.), and a set of elementary connections representing a vehicle that connects two stations without stops in between. All the above optimization problems exist in two flavors: the *basic* and the *realistic* one [26]. The latter introduces some additional constraints to take into account the time required by a passenger for moving from one vehicle to another one within a station (*transfer time*). In this paper we focus only on realistic models.

The models proposed in the literature to solve such problems can be broadly classified into two categories: those representing the timetable as an *array*, and those representing it as a *graph* [2]. Two of the most successful examples of the array-based model are the Connection Scan Algorithm (CSA) [15] and the Round-bAsed Public Transit Optimized Router (RAPTOR) [13]. CSA exploits the acyclic nature of some timetables to solve the earliest arrival problem. In CSA all the elementary connections of a timetable are stored in a single array which is scanned only once for each query. In RAPTOR the timetable is stored as a set of arrays of trips and routes which are used by a dynamic programming algorithm to solve the multi-criteria problem. The graph-based models store the timetable as a suitable graph and execute a shortest path algorithm to compute an optimal route. There exist two main approaches: the *time-expanded* and the *time-dependent* model [26]. The former model explicitly represents each time event (departure or arrival) in the timetable as a node. The arcs represent elementary connections between two events or waiting within stations, and their weights usually represent the time difference between the corresponding events. The latter model represents each station as a node and there is an arc between two nodes if there exists at least one elementary connection between the two stations represented by such nodes. The weight of an arc is time-dependent, i.e., it is a function that depends on the time at which a particular arc is scanned during the shortest path search. The time-expanded model produces a graph with a larger number of nodes and arcs and thus larger query times. A variant of the realistic time-expanded model having a smaller number of nodes and arcs (the so called *reduced time-expanded model*) has been proposed in [26].

From experimental results, it turns out that the approaches based on array representation are faster than those based on graphs [2, 13, 15]. Nevertheless, during the last years, a great research effort has been devoted to devise many so-called *speed-up techniques* which heuristically speed up the Dijkstra's algorithm for shortest paths (see [2, 4]). These techniques are mainly focused on finding optimal routes on *road networks* where they exhibit a huge speed-up factor over the basic Dijkstra's algorithm. Therefore, a promising approach could be that of adapting the speed-up techniques devised for road networks to timetable graphs [5, 12]. Following this direction, a modification of the realistic time-expanded model has been proposed and shown to harmonize well with several known speed-up techniques [12]. However, the graph models are not suitable to incorporate dynamic changes in the timetable. In fact, if the time duration of some connections changes (due to, e.g., the delay of a train), the graphs do not properly represent the modified timetable and hence the computed route could be not optimal or even not feasible. As an example, in a case study for the public transport system of Rome it has been shown that exploiting the published timetable does not lead to optimal or nearly-optimal routes [18]. Updating the graphs according to the modification in the timetable is time-consuming and in many cases it requires *topological changes* of the graph (i.e., arc or node additions and deletions) [11]. Moreover, the above mentioned speed-up techniques are not able to handle possible changes in the timetable. This is due to the fact that most of them are based on the pre-computation of additional information that are later exploited to answer queries. When a timetable modification occurs, the preprocessed information are no longer reliable and must be re-computed from-scratch, usually requiring a long computational time. Of particular impact are again the topological changes in the graph. The dynamic behavior of Transfer Patterns, a speed-up technique specifically developed for public transformation system [1], has been studied in [3]. It is shown that without performing the preprocessing from-scratch that technique gives optimal results for the vast majority (but not for all) of the queries. An online problem where delays are continuously reported to the journey planner has been studied in [25]. Regarding array-based models, RAPTOR is able to handle dynamic changes of the timetable since it is not based on preprocessing. Moreover, some dynamic speed-up techniques have been proposed for road networks which allow to handle dynamic updates [6, 9, 10, 14, 16, 28, 29].

This work aims at improving the performance of graph-based models under dynamic changes in timetable information systems. Our contributions are threefold.

First, we focus on the realistic and reduced time-expanded models by providing a simplified and optimized version of the update routine of [11]. This new routine is used in combination with the dynamic graph structure of [24] which is able to efficiently handle topological changes. Furthermore, we heuristically improve the query algorithm for the time-expanded models, which significantly improves its query time.

Second, we propose a new graph-based model for representing timetable information, called *dynamic timetable model* (dynTM), that reduces the number of changes needed in the graph as a consequence of a timetable modification. Model dynTM does not require any topological change and updates only few arc weights. At the same time, dynTM is not based on time-dependent arc-weights, thus allowing to easily incorporate realistic constraints. Moreover, dynTM produces a smaller number of nodes and arcs compared to the realistic and the reduced time-expanded models [26], and therefore, a smaller query time.

Both the above models are based on graph representations and therefore they are suitable for combination and adaptation with known speed-up techniques. To demonstrate this fact, we show how to adapt the unidirectional ALT algorithm [20] to such models. We have chosen ALT since it supports dynamic changes [10] and since a careful implementation of it can boost its performance [17].

Third, we conducted a comparative experimental study of all these implementations on several long-distance and local European public transportation timetables. Regarding the update time our study shows that both models require negligible update time after the occurrence of a delay (order of microseconds). In particular, the time required by dynTM is always the smallest one. Regarding the query time, the heuristic query algorithm for the reduced time-expanded model combined with ALT outperforms the other methods and needs a computational time that is comparable to that required by some array-based models. Finally the experiments confirm that the space required by dynTM is smaller than that required by the other models. Table 1 reports indicative results w.r.t. update time, query time, and graph size achieved with the local public transportation system of London made of about 14M of elementary connections. More results are presented in Section 6.

2 Preliminaries

A *timetable* consists of data concerning: stations, trains (or other means of transportation) connecting stations, and departure and arrival times of trains at stations. More formally,

Table 1 Results for the public transportation system of London ($14M$ elementary connections).

	Time-exp. (basic)	Time-exp. (heuristic)	Time-exp. (heuristic+ALT)	dynTM	dynTM (with ALT)
query (ms)	331.07	31.52	9.41	51.54	12.75
update (μs)	477.23			271.46	
graph size	$n = 28M$, $m = 55M$			$n = 14M$, $m = 42M$	

a timetable $\mathcal{T}$ is defined by a triple $\mathcal{T} = (\mathcal{Z}, \mathcal{B}, \mathcal{C})$, where $\mathcal{Z}$ is a set of trains, $\mathcal{B}$ is a set of stations, and $\mathcal{C}$ is a set of *elementary connections* whose elements are 5-tuples of the form $c = (Z, S_d, S_a, t_d, t_a)$. Such a tuple is interpreted as train $Z \in \mathcal{Z}$ leaves station $S_d \in \mathcal{B}$ at time t_d, and the immediately next stop of train Z is station $S_a \in \mathcal{B}$ at time t_a. If x denotes a tuple's field, then the notation $x(c)$ specifies the value of x in the elementary connection c (e.g., $t_d(c)$ denotes the departure time in c). The departure and arrival times $t_d(c)$ and $t_a(c)$ of an elementary connection c within a day are integers in the interval $\{0, 1, \ldots, 1439\}$ representing time in minutes after midnight. We assume that $|\mathcal{C}| \geq \max\{|\mathcal{B}|, |\mathcal{Z}|\}$, as we do not consider trains and stations that do not take part to any connection.

Given two time instants t_1, t_2, we denote by $\Delta(t_1, t_2)$ the time that passes between them, assuming that t_2 occurs after t_1, i.e $\Delta(t_1, t_2) = t_2 - t_1 (\text{mod } 1440)$. The *length* of an elementary connection c, denoted by $\Delta(c)$, is the time that passes between the departure and the arrival times of c assuming that c lasts for less than 24 hours, i.e $\Delta(c) = \Delta(t_d(c), t_a(c))$.

Given an elementary connection c_1 arriving at station S and an elementary connection c_2 departing from the same station S, if $Z(c_1) \neq Z(c_2)$, it is possible to transfer from $Z(c_1)$ to $Z(c_2)$ only if the time between the arrival and the departure at station S is larger than or equal to a given, *minimum transfer time*, denoted by *transfer*(S). We assume that *transfer*$(S) < 1440$, for each $S \in \mathcal{B}$. An *itinerary* in a timetable $\mathcal{T}$ is a sequence of elementary connections $P = (c_1, c_2, \ldots, c_k)$ such that, for each $i = 2, 3, \ldots, k$, $S_a(c_{i-1}) = S_d(c_i)$ and

$$\Delta(t_a(c_{i-1}), t_d(c_i)) \geq \begin{cases} 0 & \text{if } Z(c_{i-1}) = Z(c_i) \\ \mathit{transfer}(S_a(c_{i-1})) & \text{otherwise.} \end{cases}$$

We say that the itinerary starts from station $S_d(c_1)$ at time $t_d(c_1)$ and arrives at station $S_a(c_k)$ at time $t_a(c_k)$. The *length* $\Delta(P)$ of an itinerary P is given by the sum of the lengths of its elementary connections, $\Delta(P) = \sum_{i=1}^{k} \Delta(c_i)$.

A *timetable query* is defined by a triple (S, T, t_S) where $S \in \mathcal{B}$ is a departure station, $T \in \mathcal{B}$ is an arrival station and t_S is a minimum departure time. There are two natural optimization criteria that are used to answer to a timetable query. They consist in finding an itinerary from S to T which starts at a time after t_S with either the minimum arrival time or the minimum number of train transfers. Such two criteria define the following two optimization problems ([26]):

- The *Earliest Arrival Problem (EAP)* is the problem of finding an itinerary from S to T which starts at a time after t_S and has the minimum length. We assume that $\Delta(P) < 1440$ for any minimum-length itinerary P.
- The *Minimum Number of Transfers Problem (MNTP)* is the problem of finding an itinerary from S to T which starts at a time after t_S and has as few transfers from a train to another one as possible.

3 The Realistic Time-Expanded Model

In the realistic time-expanded model [26] a timetable is modeled as a directed graph, the *realistic time-expanded graph*, as follows: for each elementary connection one *departure* and

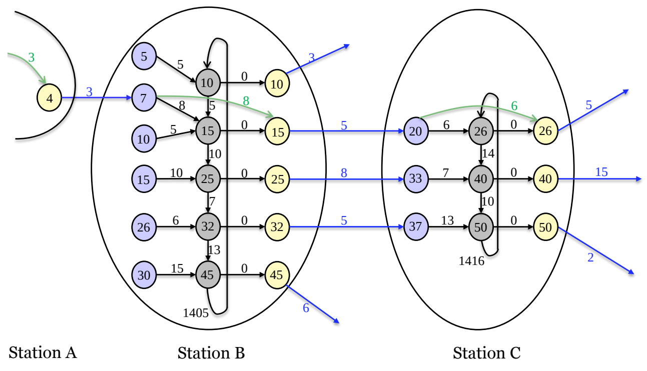


Figure 1 Arrival, transfer, and departure nodes are drawn in blue, gray and yellow, respectively. Connection and arrival-departure arcs are drawn in blue and green, respectively. Arcs with at least one transfer node endpoint are drawn in black. The minimum transfer time is 5 mins.

one *arrival* node are created and a *connection* arc is inserted between them. For each departure event, one *transfer* node is created which connects to the respective departure node by a *transfer-departure* arc having weight 0. This is done to model transfers within stations. Given a node u, $t(u)$ denotes the time-stamp of u with respect to the original timetable. To ensure a minimum transfer time at a station S, an *arrival-transfer* arc from each arrival node u is inserted to the smallest (considering time) transfer node v such that $\Delta(t(u), t(v)) \geq transfer(S)$.

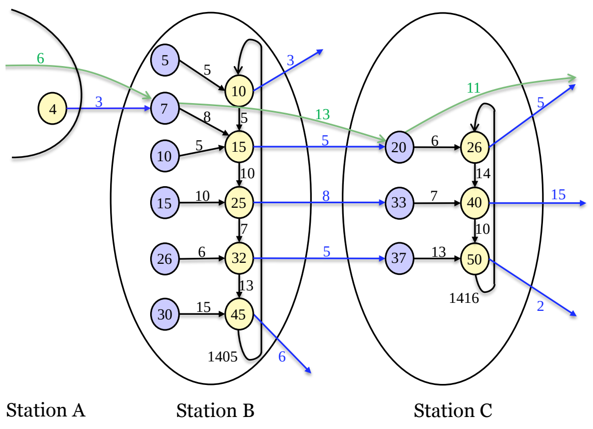
To ensure the possibility to stay in the same train when passing through a station, an additional *arrival-departure* arc is created which connects the arrival node with the appropriate departure node belonging to this same train. Further, to allow transfers to an arbitrary train, transfer nodes are ordered non-decreasing. Two adjacent nodes (w.r.t. the order) are connected by an arc from the smaller to the bigger (in terms of time) node. To allow transfers over midnight, an overnight-arc from the biggest to the smallest node is created. For each arc $e = (u, v)$ in the time-expanded graph the weight $w(e)$ is defined as the time difference $\Delta(t(u), t(v))$. Hence, for each path from a node u to another node v in the graph, the sum of the arc weights along the path is equal to the time difference $\Delta(t(u), t(v))$. Storing this graph requires $O(|\mathcal{C}|)$ space, as it has $n = 3|\mathcal{C}|$ nodes and $4|\mathcal{C}| \leq m \leq 5|\mathcal{C}|$ arcs. Figure 1 shows a realistic time-expanded graph.

Given a realistic time-expanded graph $G = (V, E)$ and a timetable query (S, T, t_S), the earliest arrival problem can be solved in the realistic time-expanded graph by finding a shortest path from s to t, where s is the transfer node with the smallest time-stamp within S such that $t(s) \geq t_S$ (or, if no such node exists, s is the node among the transfer nodes of S such that $t(s)$ is minimum), and t is an arrival node within T with minimum distance to s (i.e. the first node of T extracted from the Dijkstra's queue).

The realistic time-expanded graph can be used to solve also MNTP. In fact, it is enough to modify the weight function of the graph by setting a weight of 1 to any arc that models a transfer in a station and a weight of 0 to any other arc. In particular, the weights of all the incoming arcs of transfer nodes which come from an arrival node are set to 1.

4 The Reduced Time-Expanded Model

In order to decrease the graph size, we adopted an approach introduced in [26] called *reduced* (realistic) time-expanded model and removed the transfer nodes and the transfer-departure



■ **Figure 2** Arrival nodes are drawn in blue while departure nodes, ordered by departure time, are drawn in yellow. Arrival nodes are now connected directly to departure nodes.

arcs. Departure nodes are merged with their corresponding transfer nodes, and arrival nodes are connected directly to departure nodes. Also arrival-departure arcs of the original realistic time-expanded graph connect now arrival nodes of neighboring stations such that they belong to the same train route. This results in a reduction in the graph size of $|\mathcal{C}|$ nodes and $|\mathcal{C}|$ arcs, and therefore in a shorter traversal time within the graph. Figure 2 shows the reduced time-expanded graph corresponding to the realistic time-expanded graph of Figure 1.

Timetable queries. We propose a variant of the Dijkstra's algorithm which exploits the model structure and some restrictions to reduce the query time. In our implementation we have adopted the following approaches.

In order to reduce the size of the priority queue of the Dijkstra's algorithm, we insert in it only arrival nodes. This can be beneficial because the computation cost of the algorithm depends mostly from the priority queue updates and these in turn from queue size. To preserve the correctness of the algorithm, only for the case of departure nodes, we extent the arc relaxation depth to 2. That is, if during an arc relaxation, the head of the arc is a departure node and its distance label is updated, we also relax its outgoing arcs by exploiting a suitably defined acyclic component between neighboring stations (for example, in Figure 2, the subgraph induced by arrival node 7 and departure nodes 15, 25, 32 in station B and arrival nodes 20, 33, 37 in station C). In order to reduce the search space, we also restrict the node exploration criteria. In more detail, taking as a reference the source station s_S, we keep track of the earliest arrival time to the currently reached stations by the algorithm. In general, between two adjacent connected stations, s_A and s_B there may be many multiple routes, at different departure times from s_A. The purpose is to exclude, from early on, the non-optimal ones. To achieve this, we can exploit that the departure nodes are sorted by increasing time. Therefore, from this point onwards, we can skip the successor departure nodes with departure time higher than the reached minimum arrival time to s_B plus (as offset for the realistic model case) the transit time of s_B. Obviously, the skipped departure nodes cannot provide an earliest arrival time to s_B. These approaches can be even independent or compatible with other speed-up techniques (such as ALT).

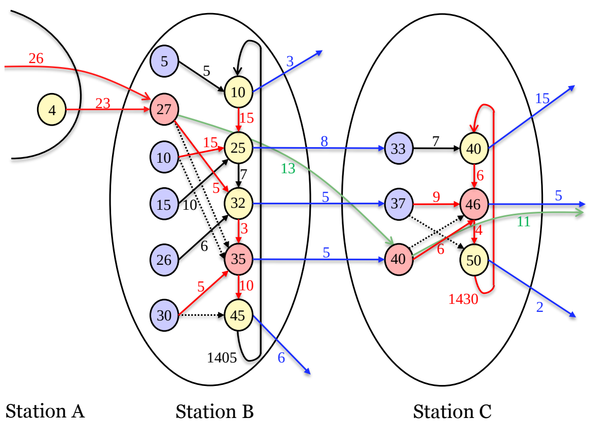


Figure 3 The arcs, departure and arrival nodes of the delayed train that need to be updated are drawn in red. The delay is 20 mins.

Handling delays. A simple approach for handling delays in the time-expanded model was proposed in [11]. When a train is delayed, the arcs of the time-expanded model within the affected stations have to be updated. The update routine consists of three steps: (i) Update the weight of the connection arc corresponding to the incoming delayed train. (ii) Update the weights of arrival-transfer and transfer-departure arcs at *all* subsequent stations through which the delayed train passes. (iii) Check for every updated arrival-transfer arc whether the update still yields valid transfer times, i.e., the arc weight is still bigger than the transfer time for this station; if not, then the arc has to be re-wired.

In this work, we have engineered, simplified, and optimized the above update routine for the case of the reduced time-expanded graph, as follows. When an update is performed, we reorder the departure nodes, in ascending order of (departure) time. Depending on the magnitude of the delay, there can be at least one arrival node that should be linked with a new earliest departure node. This requires a modification on the topology of the realistic time-expanded graph, in order to remain valid. The affected arcs are those having tail the delayed arrival node, head the delayed departure node (if the train continues its travel to another station) and head the new successor departure node of the delayed departure node. To maintain the invariant of keeping the departure nodes ordered according to time, we have to move the delayed departure node in its proper position (in a way similar to moving a node from one location to another in a linked list), and then we only need to link the arrival tail nodes with the proper earlier departure nodes so that transfer times within the station are respected. Alongside we update the arcs with the new correct weights. This operation requires only changing the node pointers of the arcs and the weights, which minimizes the update cost, and in contrast to the original approach it keeps the number of the arcs constant. The only disadvantage is that the departure nodes may now be not optimally sorted within the memory blocks and hence deteriorate the locality of references. In order to reduce the consequent impact on the performance of the query time, we initially group and pack together in memory all the departure nodes for each station. Preliminary experiments showed that the new (simplified and optimized) routine is at least 50% faster than the original one. Figure 3 illustrates the execution of the aforementioned algorithm, on the reduced time-expanded graph of Figure 2, in case of a 20 minutes delay.

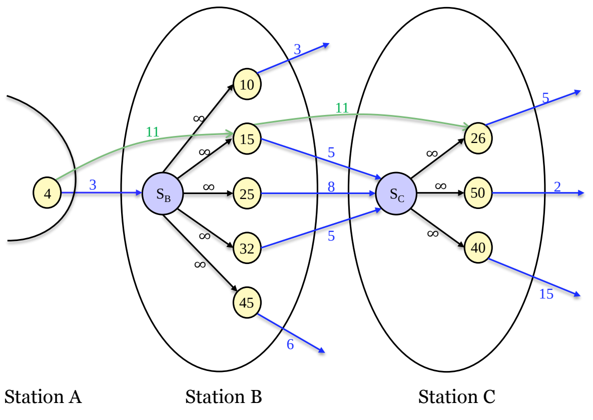


Figure 4 Switch nodes are drawn in blue while departure nodes, ordered by arrival time, are drawn in yellow. Inside each departure node the departure time of the corresponding elementary connection is reported. Connection arcs are drawn in blue, switch arcs are drawn in black while train arcs are drawn in green.

5 The Dynamic Timetable Model

In this section, we describe our new approach, called *dynamic timetable model* (dynTM for short), to solve the EAP and the MNTP problems.

Timetable model Given $\mathcal{T} = (\mathcal{Z}, \mathcal{B}, \mathcal{C})$, we define a directed graph $G = (V, E)$ called *dynamic timetable graph* and a weight function $w : E \to \mathbb{N}$ as follows.

- For each station S in $\mathcal{B}$, a node s_S, called *switch node* of S, is added to V;
- For each elementary connection $c = (Z, S_d, S_a, t_d, t_a) \in \mathcal{C}$ a node d_c, called *departure node* of c, is added to V and an arc (d_c, s_{S_a}) of c, called *connection arc*, connecting d_c to the switch node s_{S_a} of S_a is added to E;
- For each elementary connection $c = (Z, S_d, S_a, t_d, t_a) \in \mathcal{C}$ an arc (s_{S_d}, d_c), called *switch arc*, connecting the switch node s_{S_d} of the departure station S_d to the departure node d_c of c is added to E;
- For each train $Z \in \mathcal{Z}$ which travels through the itinerary $(c_1, c_2, \ldots, c_k)$, an arc, called *train arc*, connecting the departure node d_{c_i} of c_i with the departure node $d_{c_{i+1}}$ of c_{i+1} is added to E, for each $i = 1, 2, \ldots, k-1$.

For each connection arc (d_c, s_{S_a}), $w(d_c, s_{S_a}) = \Delta(t_a(c), t_d(c))$. For each train arc $(d_{c_i}, d_{c_{i+1}})$, $w(d_{c_i}, d_{c_{i+1}}) = \Delta(t_d(c_i), t_d(c_{i+1}))$. The weight of each switch arc is set to a default infinity value. Moreover, for each switch node s_S, we maintain the station S it is associated with and for each departure node d_c, we maintain the departure time $t_d(c)$ and the train $Z(c)$ of connection c which d_c is associated with. Figure 4 shows the dynamic timetable graph corresponding to the realistic time-expanded graph of Figure 1.

The graph is stored by using a forward-star representation where, for each switch node s_S, the switch arcs (s_S, d_c) outgoing from s_S are sorted according to the arrival time $t_a(c)$ of the elementary connection c associated with node d_c, in non-decreasing order.

The above data structure requires $O(|\mathcal{C}|)$ space as it needs to store a graph with $n = |\mathcal{B}| + |\mathcal{C}|$ nodes and $m \leq 3|\mathcal{C}|$ arcs. The additional information requires $O(|\mathcal{B}|)$ space for the station stored at each switch node and $O(|\mathcal{C}|)$ space for the information stored at each departure node. We recall that $|\mathcal{C}| \geq \max\{|\mathcal{B}|, |\mathcal{Z}|\}$.

Timetable queries. An EAP query (S, T, t_S) is answered by executing a modified Dijkstra's algorithm in G starting from the switch node s_S of S.

We use a vector of flags D_S for each switch node s_S. The size of D_S is given by the number of stations S' such that there exists an elementary connection departing from S and arriving at S'. We denote the element of D_S associated to S' as $D_S[S']$. Initially, all the flags of D_S are set to false, for each $S \in \mathcal{B}$.

When a switch node s_A is inserted or decreased in the Dijkstra's queue during a relaxation step, the algorithm maintains, along with the distance to s_A, also the connection c' such that the arc $(d_{c'}, s_A)$ is the one that has been relaxed. We assume that the switch node s_S of the departure station S is inserted in the queue at the initialization step with distance 0 and connection c' such that $t_d(c') + w(d_{c'}, s_S) = t_S$. Moreover we set $\mathit{transfer}(S) = 0$.

Let us consider the time when a switch node s_A, associated with station $A \in \mathcal{B}$, is extracted from the Dijkstra's queue. Let $\mathit{dist}(s_S, s_A)$ be the distance from s_S to s_A extracted from the queue and let c' be the elementary connection associated with $\mathit{dist}(s_S, s_A)$. The value of $\mathit{dist}(s_S, s_A)$ corresponds to the minimum time required to reach station A from station S, departing at time t_S. The algorithm first computes the value $x = t_d(c') + w(d_{c'}, s_A) (\text{mod } 1440)$, which represents the arrival time of connection c'. Then, for each switch arc (s_A, d_c) (i.e. for each elementary connection c such that $S_d(c) = A$), it compares x with $t_d(c)$ and *enables* the arc (s_A, d_c) if $D_S[S_a(c)] = \mathit{false}$ and

$$\Delta(x, t_d(c)) = t_d(c) - x (\text{mod } 1440) \geq \begin{cases} 0 & \text{if } Z(c) = Z(c') \\ \mathit{transfer}(A) & \text{otherwise.} \end{cases} \tag{1}$$

The arc (s_A, d_c) is enabled by setting $w(s_A, d_c)$ to $\Delta(x, t_d(c))$.

The switch arcs (s_A, d_c) are scanned according to their ordering in the forward star representation (that is according to the arrival time $t_a(c)$), starting from the first arc such that $t_d(c) \geq x$. If (s_A, d_c) is the first arc to be enabled w.r.t. some station $S' = S_a(c)$ (i.e. the one with the smallest arrival time), then the value of $D_A[S']$ is set to *true* when the first arc $(s_A, d_{c'})$ such that $S_a(c') = S'$ and $\Delta(t_a(c), t_a(c'))(\text{mod } 1440) > \mathit{transfer}(S')$ is scanned. The time instants $t_a(c)$ and $t_a(c')$ can be computed by using the value of x, $t_d(c)$ and $t_d(c')$ and the arc weights. The scanning of switch arcs of a station A is stopped when the vector D_A has only true elements and the Dijkstra's search is then pruned.

Therefore, if two switch arcs (s_A, d_{c_1}) and (s_A, d_{c_2}) (corresponding to two elementary connections c_1 and c_2) lead to the same station B, fulfill Inequality 1, and have two arrival times that differ for a value greater than $\mathit{transfer}(B)$, then only the one with smallest arrival time is enabled. In other words, if $x \leq \min\{t_d(c_1), t_d(c_2)\}$ and $t_a(c_1) < t_a(c_2) + \mathit{transfer}(B)(\text{mod } 1440)$, then $w(s_A, d_{c_1}) = t_d(c_1) - x$ and $w(s_A, d_{c_2}) = \infty$ ties are broken arbitrarily. If we assume that $t_a(c_2)$ is the smallest arrival time that fulfills the above condition, then the value of $D_A[B]$ is set to *true* when arc (s_A, d_{c_2}) is scanned.

Note that, the above behavior is performed also for the switch node s_S of the departure station S, given the initialization values of the queue. The Dijkstra's search is stopped as soon as the switch node s_T associated to the arrival station T is extracted from the queue and the arrival time t_T is given by $\mathit{dist}(s_S, s_T)$.

The proof of the following theorem will be given in the full paper.

▶ **Theorem 1.** *The modified Dijkstra's algorithm solves EAP in* $O(|\mathcal{C}| \log |\mathcal{C}|)$ *time.*

An MNTP query (S, T, t_S) can be solved similarly to an EAP one. The only differences are: (i) We do not use vector D and then all the switch arcs that satisfy transfer time

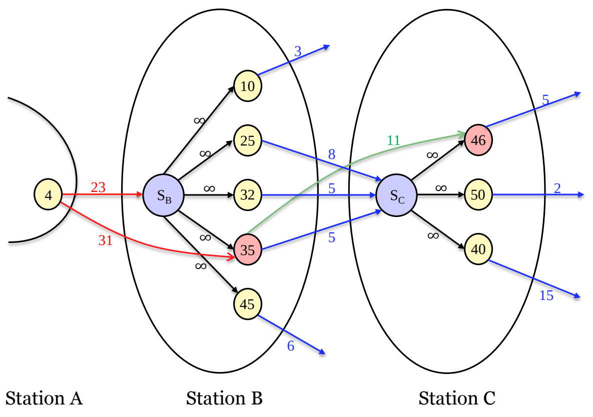


Figure 5 A delay of 20 minutes induces two arc weight changes and the update of the time associated to the corresponding departure nodes (red nodes).

constraints (Inequality 1) are enabled and (ii) when a switch node s_A is extracted from the queue with associated connection c', the weight of each switch arc (s_A, d_c) is set to 0, if $Z(c) = Z(c')$, and to 1 otherwise.

Handling delays. Let us assume that we are given a timetable $\mathcal{T}$ represented as above and a delay occurs on a connection c. The delay is modelled as an increase of d minutes on the arrival time, $t'_a(c) = t_a(c) + d(\text{mod } 1440)$. The timetable is then updated according to some specific policy which depends on the network infrastructure. The obtained timetable is called *disposition timetable* $\mathcal{T}'$ and it differs from $\mathcal{T}$ for the arrival and departure times of the trains that depend on $Z(c)$ in $\mathcal{T}$ (see e.g. [7, 8, 19, 23, 27] for examples of policies used to update a timetable).

In our model, it is enough to update the time associated to the departure node $d_{c'}$, the weight of the connection arc $(d_{c'}, S_a(c'))$, and the weight of the train arc $(d_{c'}, d_{c''})$, for each connection c' that changed from $\mathcal{T}$ to $\mathcal{T}'$. This can be done in linear time by performing a graph search on G starting from the departure node d_c associated with c. In the case that G is used to answer to EAP queries some further computation is needed as the array representing the arcs must be sorted according to the new values of the arrival times. This can be done in $O(|\mathcal{C}| \log |\mathcal{C}|)$ time as, if m_i denotes the number of nodes outgoing from each switch node s_i, then $\sum_{i \in \mathcal{B}} m_i \leq m$ and hence the overall time is given by $O(\sum_{i \in \mathcal{B}} m_i \log(m_i)) = O(\log m \sum_{i \in \mathcal{B}} m_i) = O(m \log m) = O(|\mathcal{C}| \log |\mathcal{C}|)$. Hence, the overall time needed to update the timetable is $O(|\mathcal{C}|)$ in the case that the model is exploited to answer to MNTP queries, and $O(|\mathcal{C}| \log |\mathcal{C}|)$ in the case that it is exploited for EAP queries. We remark that this is an upper bound which is far from being realistic as the stations that change some time references are much less than $|\mathcal{B}|$, especially thanks to robust design of timetables [7, 8, 19, 23, 27]. Figure 5 shows how dynTM handles a delay on the dynamic timetable graph of Figure 4.

In the experimental section, we assume that the policy adopted is that no train waits for a delayed one. Therefore, the only time references which are updated are those regarding the departure times of $Z(c)$. Moreover, we assume that the policy does not take into account any possible slack times and hence the time references are updated by adding $d(\text{ mod } 1440)$.

Comparison with the time-expanded models. In this section, we compare dynTM with the realistic and the reduced time-expanded models.

First, in case of delays, the time-expanded models require, after a reordering of the arrival, departure, and transit nodes, also an update (insertion/deletion) of arcs of the graph (see e.g. [11]). This behavior could imply a large computational time which depends on the way the graph is stored. On the contrary, dynTM is able to keep updated its data structure in case of delays in almost linear time and without any change in the graph topology. In fact, a delay in the timetable induces few arc weight changes and the update of the time associated to the corresponding departure nodes. Note that, this last operation can require, in some cases, a reordering step in the departure nodes of the stations involved by the change with respect to new arrival times.

Second, although dynTM and the two time-expanded models asymptotically require the same space complexity, the graph in the new model has a smaller number of nodes and arcs. In fact, the realistic time-expanded model requires $3|\mathcal{C}|$ nodes and at least $4|\mathcal{C}|$ arcs, the reduced time-expanded model requires $2|\mathcal{C}|$ nodes and at least $3|\mathcal{C}|$ arcs, while dynTM requires $|\mathcal{B}| + |\mathcal{C}|$ nodes and at most $3|\mathcal{C}|$ arcs (we recall that $|\mathcal{C}| \geq |\mathcal{B}|$). On the other hand, Dijkstra's algorithm executed in dynTM must perform the additional step of enabling arcs and computing the weights of the switch arcs.

Adapting to speed-up techniques. As already mentioned, one of the advantages of graph-based models for timetable information systems over the faster array-based ones is that the former models can exploit the so-called speed-up techniques for shortest path developed during the last years. Indeed, many of such techniques can be easily adapted to be used in combination with dynTM and thus improve the query time. To motivate this statement, in the following we show how to adapt one of the simplest speed-up techniques (namely ALT [20]) to dynTM. The idea behind the ALT algorithm is to direct the Dijkstra's search towards the target t of the query by adding a *feasible potential* to the priority of each node in the queue. In ALT, the feasible potentials are computed as follows. Given a set of nodes $L \subseteq V$ called *landmarks*, the feasible potential of a node $u \in V$ towards a target t is computed as $\pi_t(u) = \max_{\ell \in L} \max\{d(u,\ell) - d(t,\ell); d(\ell,t) - d(\ell,u)\}$. By the triangle inequality follows that $\pi_t(u)$ is a lower bound to the distance $d(u,t)$ and this is enough to prove the correctness of the shortest path algorithm (see [20] for more details).

The adaptation of ALT to the time-expanded models is pretty easy, and several variants have already been proposed, e.g. in [12].

In our approach, we select as landmarks the switch nodes, each of which represents the arrival node group of a station. Therefore the lower bound distance, $dist(s_A, s_B)$, between two switch nodes, s_A and s_B, denotes the minimum travel time, from any arrival node of station A to any arrival node of station B. These lower bound distances can be computed during the preprocessing phase by running single-source queries from each switch node. The tightest lower bounds, with this method, can be obtained by storing all pair station distances $O(|\mathcal{B}|^2)$. This makes sense particularly when the stations are relatively few in number.

We also used ALT along with the restricted node exploration, described in Section 4, for both the reduced time-expanded graphs and dynTM. The contribution of ALT is that of making the goal-directed search pulling faster towards the target station. The contribution of node pruning, instead, is that of removing several non-optimal arrivals between adjacent stations. The combination of the approaches reduces much more the search space size and leads to a more efficient algorithm.

Table 2 Tested timetables and sizes of the corresponding graphs; orig = original, red = reduced.

type	$\mathcal{T}$	$\|\mathcal{B}\|$	$\|\mathcal{C}\|$	TE (orig)		TE (red)		DynTM	
				$\|\mathbf{V}\|$	$\|\mathbf{E}\|$	$\|\mathbf{V}\|$	$\|\mathbf{E}\|$	$\|\mathbf{V}\|$	$\|\mathbf{E}\|$
train	efz	2 198	41 613	124 839	208 065	83 226	159 290	43 811	124 839
	d0i	6 493	428 982	1 286 946	2 144 910	857 964	1 668 171	435 475	1 286 946
	eur	7 786	596 129	1 788 387	2 912 210	1 192 258	2 316 081	603 915	1 719 952
bus	bts	716	12 689	38 067	63 445	25 378	49 862	13 405	38 067
	ks	1 865	44 744	134 232	223 720	89 488	175 536	46 609	134 232
	bvb	2 874	292 542	877 626	1 462 710	585 084	1 154 792	295 416	877 626
mixed	Berlin	6 113	3 887 965	11 663 895	19 439 825	4 085 900	6 128 800	2 044 637	4 586 247
	London	11 561	13 995 098	41 985 294	69 599 780	27 990 196	55 604 682	14 006 659	41 609 584

6 Experimental Analysis

In this section we report the results of our experimental study. Our experiments have been performed on a workstation equipped with an Intel Quad-core i5-2500K 3.30GHz CPU and 12GB of main memory, and our implementations were done in C++ (gcc compiler v4.6.3 with optimization level O4).

Input data and parameters. As input data to our experiments we used 3 train and bus timetables from a large data set provided by HaCon [21] for scientific use. We also used two different source timetable data sets in General Transit Feed (GTFS) format, containing various means of transportation. We built, for each timetable $\mathcal{T}$, a realistic time-expanded, a reduced time-expanded, and a dynamic timetable graph. For representing the graphs, we used a packed memory graph [24] for the time-expanded graphs, and a forward-star representation for the dynamic timetable graph. We used a binary heap when a priority queue was needed.

In Table 2 detailed information about the timetables and the corresponding graphs are reported. In particular, we report, for each timetable, the number of stations and the number of elementary connections between stations, the number of nodes and arcs of the corresponding graph for each model. Table 2 confirms the analysis reported in Sections 4 and 5, regarding the sizes of the models. In fact, for each timetable $\mathcal{T} = (\mathcal{Z}, \mathcal{B}, \mathcal{C})$, we notice that the number of nodes is exactly $3|\mathcal{C}|$, $2|\mathcal{C}|$, and $|\mathcal{B}| + |\mathcal{C}|$ while the number of arcs is always smaller than $5|\mathcal{C}|$, $4|\mathcal{C}|$, and $3|\mathcal{C}|$ for the realistic, the reduced time-expanded, and dynTM models, respectively.

Timetable queries. In order to test the performance of the three models, we carried out, for each timetable, EAP queries and evaluated the time required for answering them. For each timetable, we generated $1,000$ EAP queries between pairs of stations, randomly chosen with uniform probability distribution, and measured the time for executing, on each type of graph, the corresponding modified Dijkstra's algorithm. The used algorithms in the experiments are a) D: the proposed Dijkstra's algorithm variant and b) ALT: uni-directed ALT, both of them combined with the restricted node exploration technique.

The results of our experiments are summarized in Table 3. Since in [26] it has been shown that the reduced model is always better than the realistic model, in this table we report only the results on the reduced time-expanded model and dynTM. In particular, we report the average computational time per query for train, bus and mixed instances, respectively. We omit results concerning MNTP queries as they lead to similar analysis.

Our experiments show that: (i) combining ALT with the restricted node exploration leads to a significant speed-up in query time; (ii) dynTM query time is comparable to that of the reduced time-expanded model. Moreover, in [26] it has been shown that queries on time-dependent graphs are faster than those on time-expanded graphs by a small constant factor in the realistic setting. It follows that dynTM query time is also comparable to that of the time-dependent model. Our experiments also show that the query time of both reduced time-expanded model and dynTM is comparable to that of array-based approaches. In fact, for example, the query time of CSA is 2 ms on an instance of London with around 5 millions connections [2], while the query time of reduced time-expanded model and dynTM is 9.41 ms and 12.75 ms, respectively, on an instance of London with almost 14 millions connections.

Notice that, the overhead w.r.t. query time of dynTM is due to the fact that there are no separated arrival to departure arcs. Taking as reference the start time within the station, in order to take the next valid departure times after start time, there is a need for looking up many departure nodes, in non-increasing order of departure time. This makes each step of algorithms more expensive than its reduced time-expanded graph variant. Therefore, the arrival node contraction results in this disadvantage.

Timetable updates. As described in Section 5, our new model is able to efficiently handle dynamic updates to the timetable. Hence, in order to evaluate the performance of the updating algorithm, we performed a set of experiments as follows: for each timetable, we randomly selected 1,000 elementary connections and, for each elementary connection, we randomly generated a delay affecting the corresponding train or bus, chosen with uniform probability distribution between 1 and 360 minutes. For each change in the timetable, we ran the algorithm for updating the dynamic timetable graph and measured the average computational time and the number of arcs affected by the change, that is the number of arcs associated to the same train or bus which has experienced the delay. For the reduced time-expanded model we used the engineered, simplified and optimized version of the update algorithm in [11], presented in Section 4.

The experimental results are shown in Table 3. In this case dynTM outperforms the reduced time-expanded model w.r.t. the update time. The results confirm that the upper bound given in Section 5 for the computational time of the updating algorithm is really far from being realistic, thus making dynTM suitable to be used in practice. In fact, even in the biggest network (London), the updating algorithm requires 271.46 μs. Moreover, only few arc weights need to be changed in the original graph to keep the EAP queries correct, on average 9.1 in train timetables and 13.5 in bus timetables. This is due to the fact that the number of stations where something changes, as a consequence of a delay, is small with respect to the size of the whole set $|\mathcal{B}|$.

7 Conclusions

In this paper we studied graph-based models for timetable information systems which are able to handle dynamic updates and experimentally showed their effectiveness in terms of both query and update times.

We have shown that graph-based models can be combined with known speed-up techniques developed for road networks, by implementing ALT and showing its effectiveness. Therefore, a possible future work is that of combining other known speed-up techniques to the tested graph-based models. In this regards, the most promising ones are those based on Arc-flags [22] as they can be combined with ALT [12] and support dynamic updates [9]. Furthermore, the efficient implementation of ALT given in [17] would further improve the query

Table 3 Comparison between reduced time-expanded graphs and dynamic timetable graphs with respect to average query time, average update time and affected arcs, respectively.

type	$\mathcal{T}$	**query** (ms)				**update** (μs)			
		TE (red)		**dynTM**		**TE (red)**		**dynTM**	
		ALT	D	ALT	D	time	arcs	time	arcs
train	efz	0.43	0.97	0.97	1.27	25.31	30.9	2.25	7.2
	d0i	1.53	6.35	3.76	8.94	40.32	32.2	7.17	7.6
	eur	1.69	7.66	3.71	9.74	43.10	33.5	8.07	9.1
bus	bts	0.17	0.28	0.29	0.43	34.47	37.7	2.19	8.8
	ks	0.67	1.34	0.78	1.36	39.49	53.8	3.04	11.1
	bvb	1.56	2.67	2.92	3.89	95.98	63.5	15.17	13.5
mixed	Berlin	9.90	21.67	13.17	37.85	194.54	61.6	80.23	9.9
	London	9.41	31.52	12.75	51.54	477.23	130.4	271.46	29.0

time. Given these combinations, it would be also interesting to experimentally compare the models studied in this paper with both array-based and time-dependent approaches.

We focused on the earliest arrival problem to demonstrate the potential of the graph based models. Therefore, another possible future work could be that of tackling the multi-criteria problem.

In addition, we plan to extend the study by: (i) analyzing different types of timetable modifications corresponding to different policies for delay management; (ii) taking into account possible slack/buffer times in the timetable; (iii) considering footpaths between stations in dynTM. To this regard, we believe that footpaths can be easily incorporated to our graph based models in an efficient way.

References

1 Hannah Bast, Erik Carlsson, Arno Eigenwillig, Robert Geisberger, Chris Harrelson, Veselin Raychev, and Fabien Viger. Fast routing in very large public transportation networks using transfer patterns. In *18th Annual European Symposium on Algorithms (ESA 2010)*, volume 6346 of *LNCS*, pages 290–301. Springer, 2010.

2 Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Mueller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato Werneck. Route planning in transportation networks. Technical Report MSR-TR-2014-4, Microsoft Research, 2014.

3 Hannah Bast, Jonas Sternisko, and Sabine Storandt. Delay-Robustness of Transfer Patterns in Public Transportation Route Planning. In *13th Work. on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS)*, pages 42–54. Schloss Dagstuhl, 2013.

4 Reinhard Bauer, Daniel Delling, Peter Sanders, Dennis Schieferdecker, Dominik Schultes, and Dorothea Wagner. Combining hierarchical and goal-directed speed-up techniques for dijkstra's algorithm. *ACM J. Exp. Alg.*, 15:Article 2.3, 2010.

5 Reinhard Bauer, Daniel Delling, and Dorothea Wagner. Experimental study of speed up techniques for timetable information systems. *Networks*, 57(1):38–52, 2011.

6 F. Bruera, S. Cicerone, G. D'Angelo, G. Di Stefano, and D. Frigioni. Dynamic multi-level overlay graphs for shortest paths. *Math. Comp. Sc.*, 1(4):709–736, 2008.

7 Serafino Cicerone, Gianlorenzo D'Angelo, Gabriele Di Stefano, Daniele Frigioni, and Alfredo Navarra. Recoverable robust timetabling for single delay: Complexity and polynomial algorithms for special cases. *Journal of Combinatorial Optimization*, 18(3):229–257, 2009.

8 Serafino Cicerone, Gianlorenzo D'Angelo, Gabriele Di Stefano, Daniele Frigioni, Alfredo Navarra, Michael Schachtebeck, and Anita Schöbel. Recoverable robustness in shunting and timetabling. In *Robust and Online Large-Scale Opt.*, volume 5868 of *LNCS*, pages 28–60. Springer, 2009.

9 Gianlorenzo D'Angelo, Mattia D'Emidio, and Daniele Frigioni. Fully dynamic update of arc-flags. *Networks*, 63(3):243–259, 2014.

10 D. Delling and D. Wagner. Landmark-based routing in dynamic graphs. In *6th Work. on Experimental Algorithms*, LNCS, pages 52–65. Springer, 2007.

11 Daniel Delling, Kalliopi Giannakopoulou, Dorothea Wagner, and Christos Zaroliagis. Timetable Information Updating in Case of Delays: Modeling Issues. Technical Report ARRIVAL-TR-0133, ARRIVAL Project, 2008.

12 Daniel Delling, Thomas Pajor, and Dorothea Wagner. Engineering time-expanded graphs for faster timetable information. In *Robust and Online Large-Scale Optimization*, volume 5868 of *LNCS*, pages 182–206. Springer, 2009.

13 Daniel Delling, Thomas Pajor, and Renato F. Werneck. *Round-Based Public Transit Routing*, pages 130–140. SIAM, 2012.

14 Daniel Delling and Renato F. Werneck. Faster customization of road networks. In *12th Symp. Exp. Alg. (SEA)*, volume 7933 of *LNCS*, pages 30–42. Springer, 2013.

15 Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Intriguingly simple and fast transit routing. In *12th Symp. Exp. Alg. (SEA)*, volume 7933 of *LNCS*, pages 43–54. Springer, 2013.

16 Julian Dibbelt, Ben Strasser, and Dorothea Wagner. Customizable contraction hierarchies. In *13th Int. Symp. on Exp. Alg. (SEA)*, volume 8504 of *LNCS*, pages 271–282. Springer, 2014.

17 Alexandros Efentakis and Dieter Pfoser. Optimizing landmark-based routing and preprocessing. In *6th ACM SIGSPATIAL Int. Work. on Computational Transp. Science.* ACM, 2013.

18 Donatella Firmani, Giuseppe F. Italiano, Luigi Laura, and Federico Santaroni. Is Timetabling Routing Always Reliable for Public Transport? In *13th Work. on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS)*, pages 15–26. Schloss Dagstuhl, 2013.

19 Matteo Fischetti, Domenico Salvagnin, and Arrigo Zanette. Fast approaches to improve the robustness of a railway timetable. *Transportation Science*, 43(3):321–335, 2009.

20 A. Goldberg and C. Harrelson. Computing the shortest path: A* search meets graph theory. In *ACM-SIAM Symposium on Discrete Algorithms (SODA05)*, pages 156–165. SIAM, 2005.

21 HaCon - Ingenieurgesellschaft mbH. `http://www.hacon.de`, 2008.

22 U. Lauther. An extremely fast, exact algorithm for finding shortest paths. *Static Networks with Geographical Background*, 22:219–230, 2004.

23 Christian Liebchen, Michael Schachtebeck, Anita Schöbel, Sebastian Stiller, and André Prigge. Computing delay resistant railway timetables. *Computers & OR*, 37(5):857–868, 2010.

24 Georgia Mali, Panagiotis Michail, Andreas Paraskevopoulos, and Christos Zaroliagis. A new dynamic graph structure for large-scale transportation networks. In *8th Int. Conf. on Algorithms and Complexity (CIAC)*, volume 7878 of *LNCS*, pages 312–323. Springer, 2013.

25 Matthias Müller-Hannemann and Mathias Schnee. Efficient timetable information in the presence of delays. In *Robust and Online Large-Scale Optimization*, volume 5868 of *LNCS*, pages 249–272. Springer Berlin Heidelberg, 2009.

26 Evangelia Pyrga, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Efficient models for timetable information in public transportation systems. *ACM J Exp Alg*, 12(2.4):1–39, 2008.

27 Michael Schachtebeck and Anita Schöbel. To wait or not to wait - and who goes first? delay management with priority decisions. *Transportation Sc.*, 44(3):307–321, 2010.

28 D. Schultes and P. Sanders. Dynamic highway-node routing. In *6th Workshop on Experimental Algorithms (WEA)*, LNCS, pages 66–79. Springer, 2007.

29 Dorothea Wagner, Thomas Willhalm, and Christos D. Zaroliagis. Geometric containers for efficient shortest-path computation. *ACM J. Exp. Alg.*, 10(1.3):1–30, 2005.