Project Number 288094

eCOMPASS

eCO-friendly urban Multi-modal route PlANning Services for mobile uSers

STREP
Funded by EC, INFSO-G4(ICT for Transport) under FP7

eCOMPASS – TR – 049

Engineering a New Model for Dynamic Timetable Information Systems

Alessio Cionini, Gianlorenzo D'Angelo, Mattia D'Emidio, Daniele Frigioni, Kalliopi Giannakopoulou, Andreas Paraskevopoulos, and Christos Zaroliagis

February 2014

Engineering a New Model for Dynamic Timetable Information Systems *

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

[1] Department of Information Engineering, Computer Science and Mathematics, University of L'Aquila, Via G. Gronchi, 18, I–67100, L'Aquila, Italy. alessio.cionini@gmail.com, {mattia.demidio, daniele.frigioni}@univaq.it.
[2] Department of Mathematics and Informatics, University of Perugia, Via Vanvitelli, 1, 06123 Perugia, Italy. gianlorenzo.dangelo@dmi.unipg.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 decade to model public transport timetables in a graph theoretical framework. The aim is to represent a timetable as a graph so that an optimal route or itinerary is found by computing afterwards a shortest path in such a graph. One of the most used models is the so called time-expanded graph model. Its main drawback is that if the timetable changes (e.g., due to a delay occurring in the network), the graph needs to be topologically updated. In this paper we propose a new model to represent the timetable of a public transportation system as a graph. Its main advantage is that it can be efficiently updated after a change in the timetable. In fact, it is based on a compact representation that allows us to update the graph by changing only few arc weights and does not require any topological modification. We also conduct a comparative experimental study on real-world timetables to assess the efficiency and practical applicability of the new model against the classical and the reduced time-expanded models. The experiments show that the new model outperforms the other two models in query and update times and also in space requirements.

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 that minimizes either the traveling

* Partially supported by the EU FP7/2007-2013 (DG CONNECT.H5-Smart Cities & Sustainability), under grant agreements no. 288094 (project eCOMPASS) and no. 609026 (project MOVESMART).

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*). In order to answer to such queries, the journey planners store the vehicle (trains, buses, ferries, etc.) timetables as a graph and execute a shortest path algorithm to compute the optimal route. In the literature (see e.g., [17]) there exist two main approaches to model timetables: the *time-expanded* and the *time-dependent* model. The former model explicitly represents each time event (departure or arrival) in the timetable as a node. The arcs of the graph represent elementary connections between two events (a train that connects the two events without stops in between) 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. Both time-expanded and time-dependent models exist in two flavors: the *basic* and the *realistic* model [17]. The latter introduces some additional constraints to take into account the time required by a passenger for moving from one train to another one within a station (*transfer time*). In this paper we focus only on realistic models.

As it turns out, the time-expanded model yields a graph with a larger number of nodes and arcs and thus larger query times. On the other hand, the time-dependent model makes difficult to incorporate additional constraints such as, e.g., the transfer times. A variant of the realistic time-expanded model, called *reduced time-expanded model* has been proposed in [17], having a smaller number of nodes and arcs. All the above models are not suitable to incorporate dynamic changes in the timetables. In fact, if the time of some connections changes (due to, e.g., the delay of a train), the graphs do not properly represent the modified timetables and hence the computed route could be not optimal or even not feasible. 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) [8, 16].

In the last years, a large number of *speed-up techniques* have been devised to heuristically speed up Dijkstra's algorithm (see [1] for an overview). Most of them are based on the pre-computation of additional information that can be effectively used to answer queries. However, these techniques are mainly focused on finding optimal routes on *road networks* and adapting them to the case of graphs representing timetable information does not yield the same query speed-up [2, 9]. In particular, in [9] the authors propose a modification of the realistic time-expanded model, and give an experimental evaluation of various speed-up techniques on this new model, by showing that it harmonizes well with known speed-up techniques. In any case, the above speed-up techniques are not able to handle possible changes in the timetables. Since they are based on a preprocessing of the graph, the timetable modification can cause the complete recomputation from-scratch of the preprocessed information that usually requires

long computational time. On the other hand, some dynamic speed-up techniques have been proposed for road networks which allow to handle the dynamic weight updates [3, 6, 10, 11, 19, 20].

Our main contribution in this work is a new time-expanded 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. 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, our model yields a smaller number of nodes and arcs compared to the realistic and the reduced time-expanded models [17], and therefore, a smaller query time.

Our second contribution is an experimental assessment of the effectiveness of dynTM. Except for implementing dynTM and its algorithms, we provided new implementations of the realistic and reduced time-expanded model along with a simplified and optimized version of the update routine in [8] for handling delays using the dynamic graph structure in [15]. We conducted a comparative experimental study of all these implementations on several European public transportation timetables. Our study shows that dynTM requires negligible update time after the occurrence of a delay, which is always smaller than that required by the other models. Moreover, the time required by dynTM to answer to a timetable query is almost always smaller than that required by the realistic and reduced time-expanded models and comparable to that required by the time-dependent one. Finally the experiments confirm that the space required by dynTM is smaller than that required by the other models. We believe that the efficiency of the query phase along with the simplicity of the graph updates make our model suitable for use in combination with several dynamic speed-up techniques.

2 Preliminaries

A *timetable* consists of data concerning: stations, trains connecting stations, and departure and arrival times of trains at stations. More formally, 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

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

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 or train transfers. Such two criteria define the following two optimization problems ([17]):

- 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 Realistic time-expanded model and delay handling

In the realistic time-expanded model [17] a timetable is modeled as a directed graph, the *realistic time-expanded graph*, as follows: for each elementary connection one *departure* and 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 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.

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. In Appendix A we give an example of a timetable and the corresponding realistic time-expanded graph.

Handling delays and the reduced time-expanded model. A simple approach for handling delays in the realistic time-expanded model was proposed in [8]. When a 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. More details on the update routine are provided in Appendix A.

In this work, we have further engineered, simplified and optimized the realistic time-expanded model as well as the aforementioned update routine. In particular, we adopted a heuristic approach introduced in [17] called reduced (realistic) time-expanded model and removed the transfer nodes and the transfer-departure arcs. The arrival nodes are connected directly to the departure nodes, and the transfer arcs connect now successive (in departure time) departure nodes. This results in a reduction in the graph size by $|\mathcal{C}|$ nodes and $|\mathcal{C}|$ arcs, and therefore in a shorter traversal time within the graph.

The update routine in case of delays is engineered, simplified, and optimized as follows (see Appendix A for an illustration). 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-expended 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.

4 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. In Appendix B we show our approach by means of the same example used in the previous section.

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 infinity. 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.

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 of 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 $transfer(S) = 0$.

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 $dist(s_S, s_A)$ be the distance from s_S to s_A extracted from the queue and let c' be the elementary connection associated with $dist(s_S, s_A)$. The value of $dist(s_S, s_A)$ corresponds to the minimum time required to reach station A from station S, departing at time t_S. The algorithm, first computes the value $x = t_d(c') + w(d_{c'}, s_A) (\mathrm{mod}\ 1440)$ 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 $D_S[S_a(c)] = false$ and

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

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 $\Delta(t_a(c), t_a(c')) (\mathrm{mod}\ 1440) > transfer(S')$ 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 $transfer(B)$, 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 $t_a(c_1) < t_a(c_2) + transfer(B) (\mathrm{mod}\ 1440)$, 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 $dist(s_S, s_T)$.

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

Proof. By the construction of the graph, the path found by the above algorithm is an itinerary in $\mathcal{T}$ starting from S at time $t \geq t_S$ and arriving at T at time t_T.

We now prove that such an itinerary has minimum length. The proof mimics the correctness proof of the Dijkstra's algorithm. In particular we prove that, when a switch node s_A is extracted from the Dijkstra's queue, the value of $dist(s_S, s_A)$ is minimum, this induces a minimum-length itinerary from S to A.

Let us assume by contradiction that A is the station such that s_A is the first node which is extracted from the queue whose value of $dist(s_S, s_A)$ is not minimum. When s_A has been inserted or decreased from the queue for the last time, it was due to the relaxation of a connection arc $(t_d(c'), s_A)$ and a switch arc $(s_B, t_d(c'))$, for some station B and connection c'. The switch arc $(s_B, t_d(c'))$ is enabled only if the connection c' is the one that leads from station B to station A with minimum arrival time and that satisfy Inequality 1. Moreover, as s_A has been decreased for the last time due to the relaxation of $(t_d(c'), s_A)$, then all the other connection arcs leading to s_A correspond to longest paths. Therefore, it must be the case that when the switch node s_B of station B is extracted from the queue, $dist(s_S, s_B)$ is not minimum, a contradiction to the fact that s_A was the first extracted node with non-minimum distance.

The computational complexity is that of the Dijkstra's algorithm in a graph with $n = |\mathcal{B}| + |\mathcal{C}|$ nodes and $m \leq 3|\mathcal{C}|$ arcs. However, the enabling step slightly increases the computational complexity. In fact, it must be done for each outgoing arc of an extracted node s_A, like the relaxation in the classical Dijkstra's algorithm, and it requires to look for an element of array D_A. Therefore, the relaxation of an arc requires $O(\log |D_A|)$ time, differently from Dijkstra which requires constant time. As D_A has at most $|\mathcal{B}|$ elements, this requires at most $O(\log |\mathcal{B}|)$ time for each arc in the graph. Overall, the overhead is $O(m \log |\mathcal{B}|)$ and the time complexity is $O(nlogn + m \log |\mathcal{B}|) = O(|\mathcal{C}| \log |\mathcal{C}|)$, as $|\mathcal{B}| \leq |\mathcal{C}|$. □

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 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, that is, the new arrival time is $t'_a(c) =$

$t_a(c) + d (\text{mod } 1440)$. 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. [4, 5, 7, 12, 14, 18] 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 be realistic as the stations that change some time references are much less than $|\mathcal{B}|$, especially thanks to robust design of timetables [4, 5, 7, 12, 14, 18].

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 against the realistic and the reduced time-expanded models, showing that the dynTM outperforms both the other models from two points of view.

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. [8]). 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

type	$\mathcal{T}$	$\lvert\mathcal{B}\rvert$	$\lvert\mathcal{C}\rvert$	TE (orig)		TE (red)		DynTM	
				$\lvert\mathbf{V}\rvert$	$\lvert\mathbf{E}\rvert$	$\lvert\mathbf{V}\rvert$	$\lvert\mathbf{E}\rvert$	$\lvert\mathbf{V}\rvert$	$\lvert\mathbf{E}\rvert$
Train	i0i	6	443	1 329	1 931	886	1488	449	1 045
	a0i	18	573	1 719	2 477	1 146	1 904	591	1 335
	f0i	24	729	2 187	2 978	1 458	2 249	753	1 536
	b0i	27	3 349	10 047	15 242	6 698	11 892	3 376	8 548
	efz	2 931	41 613	124 839	201 523	83 226	159 290	44 544	119 114
	d0i	6 602	428 982	1 286 946	2 097 169	857 964	1 668 171	435 584	1 239 329
Bus	wez	54	861	2 583	4 221	1 722	3 368	915	2 556
	meg	369	3 295	9 885	16 088	6 590	12 762	3 664	9 732
	vib	177	5 983	17 949	29 546	11 966	23 586	6 160	17 633
	bts	726	12 689	38 067	62 425	25 378	49 862	13 415	37 707
	ks	1879	44 744	134 232	220 074	89 488	175 536	46 623	131 655
	bvb	2 874	292 542	877 626	1 446 935	585 084	1 154 792	295 416	865 559

Table 1: Tested timetables and sizes of the corresponding graphs; orig = original, red = reduced.

weights of the switch arcs. However, we will experimentally show that this time overhead is small with respect to the improvement in performance due to the reduced graph size.

5 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 8GB 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 six train timetables and six bus timetables from a large data set provided by HaCon [13] for scientific use. We built, for each timetable, a realistic time-expanded, a reduced time-expanded, and a dynamic timetable graph. For representing the graphs, we used a packed memory graph [15] 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 1 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 1 confirms the analysis reported in Sections 3 and 4, 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 results of our experiments are summarized in Table 2a. Since in [17] 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, in this table we report the average computational time per query for train and bus instances, respectively. We omit results concerning MNTP queries as they lead to similar analysis.

Our experiments clearly show that dynTM outperforms the reduced time-expanded model w.r.t. the query time. This implies that the time overhead induced by the additional steps performed by the modified Dijkstra's algorithm in the new model is small w.r.t. the improvement in performance due to the reduced graph size. Note that in [17] the authors show that queries on time-dependent graphs are faster than those on time-expanded graphs by a small constant factor in the realistic setting. Therefore, the query time of our model is comparable to that of the time-dependent model.

Timetable updates. As described in Section 4, 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 [8], presented in Section 3.

The experimental results are shown in Table 2b. Also 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 4 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 (d0i), the updating algorithm requires 5.7 μs. Moreover, only few arc weights need to be changed in the original graph to keep the EAP queries correct, on average 8.8 in train timetables and 14.2 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 $|\mathcal{B}|$.

References

1. R. Bauer, D. Delling, P. Sanders, D. Schieferdecker, D. Schultes, and D. Wagner. Combining hierarchical and goal-directed speed-up techniques for dijkstra's algorithm. *ACM J. Exp. Alg.*, 15:Article 2.3, 2010.

type	$\mathcal{T}$	avg query time (ms)	
		TE (red)	dynTM
Train	i0i	0.004	0.008
	a0i	0.013	0.015
	f0i	0.015	0.022
	b0i	0.026	0.036
	efz	1.886	1.231
	d0i	10.414	8.009
Bus	wez	0.029	0.031
	meg	0.121	0.102
	vib	0.078	0.092
	bts	0.446	0.382
	ks	2.178	1.241
	bvb	4.224	3.276

(a)

type	$\mathcal{T}$	avg update time (μs)		avg updated arcs	
		TE (red)	dynTM	TE (red)	dynTM
Train	i0i	0.8	0.6	3.9	1.5
	a0i	1.7	0.7	4.7	1.6
	f0i	2.1	1.0	5.3	1.7
	b0i	4.8	1.6	9.0	2.2
	efz	7.4	1.9	26.9	7.6
	d0i	19.4	5.7	32.9	8.8
Bus	wez	2.3	0.8	26.5	6.2
	meg	3.4	1.5	29.7	6.5
	vib	6.4	1.3	37.9	7.6
	bts	6.9	2.0	40.0	8.9
	ks	12.6	2.3	53.7	11.5
	bvb	50.1	11.1	69.4	14.2

(b)

Table 2: Comparison between reduced time-expanded graphs and dynamic timetable graphs with respect to average query time (a), average update time and affected arcs (b), respectively.

2. R. Bauer, D. Delling, and D. Wagner. Experimental study of speed up techniques for timetable information systems. *Networks*, 57(1):38–52, 2011.
3. F. Bruera, S. Cicerone, G. D'Angelo, G. D. Stefano, and D. Frigioni. Dynamic multi-level overlay graphs for shortest paths. *Math. Comp. Sc.*, 1(4):709–736, 2008.
4. S. Cicerone, G. D'Angelo, G. Di Stefano, D. Frigioni, and A. Navarra. Recoverable robust timetabling for single delay: Complexity and polynomial algorithms for special cases. *Journal of Combinatorial Optimization*, 18(3):229–257, 2009.
5. S. Cicerone, G. D'Angelo, G. Di Stefano, D. Frigioni, A. Navarra, M. Schachtebeck, and A. Schöbel. Recoverable robustness in shunting and timetabling. In *Robust and Online Large-Scale Opt.*, volume 5868 of *LNCS*, pages 28–60. Springer, 2009.
6. G. D'Angelo, M. D'Emidio, D. Frigioni, and C. Vitale. Fully dynamic maintenance of arc-flags in road networks. In *Proc. 11th Int. Symp. on Exp. Alg. (SEA)*, volume 7276 of *LNCS*, pages 135–147. Springer, 2012.
7. G. D'Angelo, G. Di Stefano, A. Navarra, and C. M. Pinotti. Recoverable robust timetables: an algorithmic approach on trees. *IEEE Tr Comp*, 60(3):433–446, 2011.
8. D. Delling, K. Giannakopoulou, D. Wagner, and C. Zaroliagis. Timetable Information Updating in Case of Delays: Modeling Issues. Technical Report ARRIVAL-TR-0133, ARRIVAL Project, 2008.
9. D. Delling, T. Pajor, and D. 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.
10. D. Delling and D. Wagner. Landmark-based routing in dynamic graphs. In *Proc. 6th Work. on Experimental Algorithms*, LNCS, pages 52–65. Springer, 2007.
11. D. Delling and R. F. Werneck. Faster customization of road networks. In *Proc. 12th Symp. Exp. Alg. (SEA)*, volume 7933 of *LNCS*, pages 30–42. Springer, 2013.
12. M. Fischetti, D. Salvagnin, and A. Zanette. Fast approaches to improve the robustness of a railway timetable. *Transportation Science*, 43(3):321–335, 2009.
13. HaCon - Ingenieurgesellschaft mbH. http://www.hacon.de, 2008.
14. C. Liebchen, M. Schachtebeck, A. Schöbel, S. Stiller, and A. Prigge. Computing delay resistant railway timetables. *Computers & OR*, 37(5):857–868, 2010.
15. G. Mali, P. Michail, A. Paraskevopoulos, and C. 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.

16. M. Müller-Hannemann and M. 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.
17. E. Pyrga, F. Schulz, D. Wagner, and C. Zaroliagis. Efficient models for timetable information in public transportation systems. *ACM J Exp Alg*, 12(2.4):1–39, 2008.
18. M. Schachtebeck and A. Schöbel. To wait or not to wait - and who goes first? delay management with priority decisions. *Transportation Sc.*, 44(3):307–321, 2010.
19. D. Schultes and P. Sanders. Dynamic highway-node routing. In *Proc. 6th Workshop on Experimental Algorithms (WEA)*, LNCS, pages 66–79. Springer, 2007.
20. D. Wagner, T. Willhalm, and C. D. Zaroliagis. Geometric containers for efficient shortest-path computation. *ACM J. Exp. Alg.*, 10(1.3):1–30, 2005.

Appendix

A The realistic time-expanded model and delay handling

In Table 3 we show a part of a timetable with 7 elementary connections, involving three stations and five trains. We report, for each elementary connection, departure and arrival stations and times, respectively. We also report the transfer time associated with the arrival station of each train. We omit part of the timetable for the sake of simplicity.

Dep. Station	Arr. Station	Dep. Time	Arr. Time	TrainID	Transfer Time
A	B	00:00	00:20	α	3
A	B	00:00	00:45	β	3
B	C	00:22	00:57	α	5
C	·	00:58	01:10	α	·
C	·	01:28	01:43	γ	·
B	·	00:47	00:59	θ	·
B	·	00:55	01:10	ϕ	·

Table 3: An example of train timetable.

The time-expanded model explicitly represents each event occurring at a station. Each event represents the departure or the arrival of a station. In order to incorporate transfer times, transfer nodes are introduced as well. In Figure 1 the time-expanded graph modeling the timetable of Table 3 is given. As a result, 4 types of arcs exist in the realistic model: connection, transfer–departure, arrival–departure, and arrival–transfer arcs.

- The connection arcs are inserted between departure and arrival events and correspond to a real connection between the stations. The weight of these arcs is the real travel time of this connection.
- For each departure event, a transfer event is inserted to the graph. The according transfer–departure arcs have a weight of 0.
- As long as a train does not end at the specific station, an arrival–departure arc is inserted between the arrival of this train in this station and its departure. Due to the fact that passengers may stay in the train, this departure event is the only one that can be reached without entering the station.
- In order to change trains at a station the train has to be left. Thus, from each arrival node an arrival–transfer arc to the next reachable transfer node (i.e., to a node corresponding to the first departure that a passenger can reach) is inserted.

We now show how the time-expanded graph has to be updated when a train is delayed by presenting the approach in [8]. When a train is delayed, it is not sufficient to simply increase the travel time of the delayed train in the time-expanded model. Instead, we have to update arcs within stations as well. As a consequence, the update routine consists of three steps.

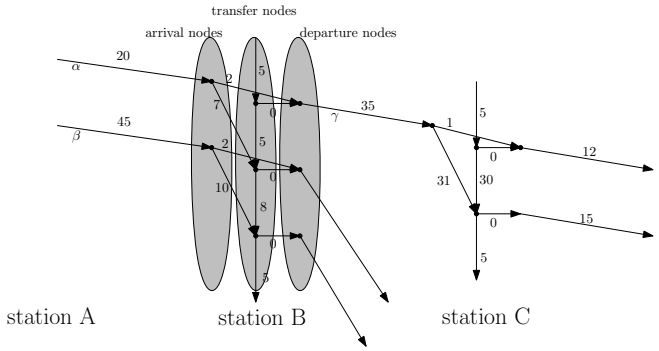

Fig. 1: The realistic time-expanded graph modelling the timetable of Table 3: at each station three types of nodes exist. The train represented by the connections α and γ stays for 2 minutes in station B. The transfer time T_B at station B is set to 3 minutes.

Step 1 (Increase Connection Weight). We identify the delayed connection and increase the weight of *only* this particular connection arc. Any other connection arc stays untouched.

Step 2 (Update Station Arcs). For all subsequent stations the delayed train stops we have to update both transfer–departure and arrival–transfer arcs. For the former we simply increase its weight from 0 to the delay Δ, while for the latter we *decrease* each arrival–transfer arc by Δ.

Step 3 (Validate Station Arcs). Our final step checks for every altered arrival–transfer arc, whether these arcs are still *valid*, i.e., the arc weight is still bigger than the transfer time for this station. In case an arc is valid, we are done for this arc, but in case not, we have to *re-wire* the arc. The target node has to be changed to the next reachable transfer node, i.e., the first node resulting in a valid arrival–transfer arc.

Figure 2 gives an example for the result of this update routine in case of a delay $\Delta = 8$ minutes.

Figures 3 and 4 illustrate the execution of the simplified and optimized update algorithm (presented in Section 3) on the reduced time-expanded graph when a delay of 20 minutes occurs.

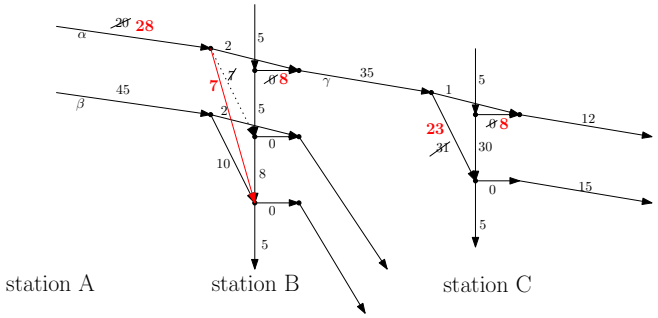

Fig. 2: Modeling delays in the time-expanded model. The graph shows the same example from Fig. 1 but train α is delayed by 8 minutes. The equivalent arc is increased by 8 minutes and the according arrival–transfer and transfer–departure arcs are altered at all following stations. For station B, we have to remove the original arrival–transfer arc and add a new one, while for station C, it is sufficient to decrease the arc weight to 23.

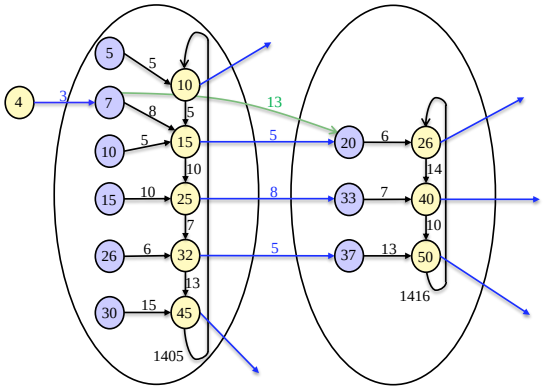

Fig. 3: Arrival nodes are drawn in blue while departure nodes, ordered by departure time, are drawn in yellow. Two stations are projected. The departure and arrival nodes of a station are enclosed within an oval outline. The minimum transfer time is 5mins.

B The dynamic timetable model and delay handling

In Figure 5, the dynamic timetable graph modeling the timetable of Table 3 is shown. Figure 6 shows how dynTM handles a delay in the timetable as described in Section 4.

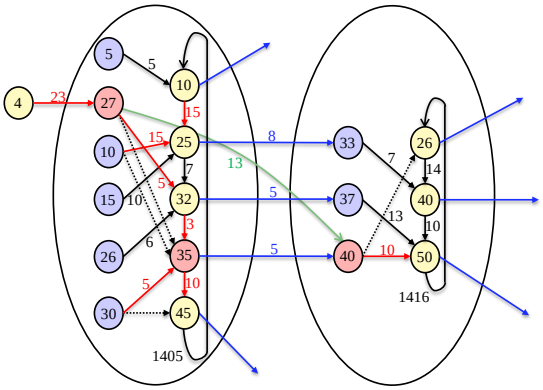

Fig. 4: The arcs, departure and arrival nodes of the delayed train, which need to be updated, are drawn in red. The delay is 20 mins.

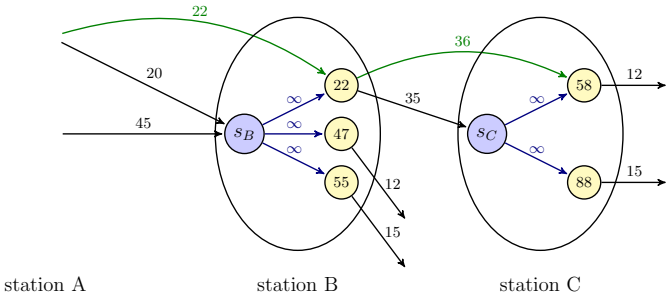

Fig. 5: The dynamic timetable graph modelling the timetable of Table 3: switch nodes are drawn in blue while departure nodes, ordered by arrival time, are drawn in yellow. Inside each departure node the departure time of the corresponding elementary connection is drawn. Connection arcs are drawn in black, while switch arcs are drawn in blue.

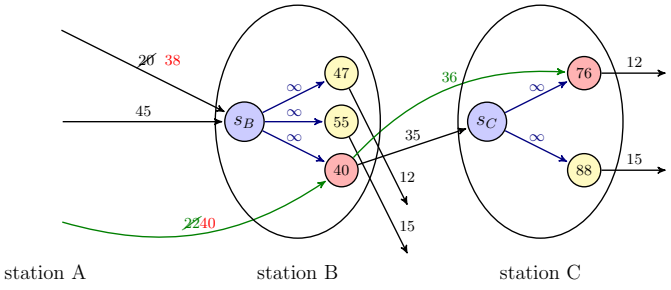

Fig. 6: Handling delays in a dynamic timetable graph: a delay of 18 minutes of train α in the timetable of Table 3 induces two arc weight changes and the update of the time associated to the corresponding departure nodes (red nodes).