Journal of Discrete Algorithms 46–47 (2017) 40–58

Contents lists available at ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

Engineering graph-based models for dynamic timetable information systems ☆

Alessio Cionini [a], Gianlorenzo D'Angelo [b], Mattia D'Emidio [b], Daniele Frigioni [a], Kalliopi Giannakopoulou [c,d], Andreas Paraskevopoulos [c,d], Christos Zaroliagis [c,d,*]

[a] Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica, Università degli Studi dell'Aquila, Via Vetoio, I-67100 L'Aquila, Italy
[b] Gran Sasso Science Institute (GSSI), Viale Francesco Crispi, I-67100 L'Aquila, Italy
[c] Department of Computer Engineering and Informatics, University of Patras, 26504 Patras, Greece
[d] Computer Technology Institute and Press "Diophantus", 26504 Patras, Greece

A R T I C L E   I N F O

Article history:
Received 23 October 2015
Received in revised form 19 March 2017
Accepted 29 September 2017
Available online 10 October 2017

Keywords:
Journey planning
Public transportation systems
Timetable models
Shortest paths
Experimental evaluation
Dynamic updates

A B S T R A C T

In the last years we have witnessed remarkable progress in providing efficient algorithmic solutions to the problem of computing best journeys (or routes) in schedule-based public transportation systems. We have now models to represent timetables that allow us to answer queries for optimal journeys in a few milliseconds, also at a very large scale. Such models can be classified into two types: those representing the timetable as an array, and those representing it as a graph. Array-based models have been shown to be very effective in terms of query time, while graph-based ones usually answer queries by computing shortest paths, and hence they are suitable to be combined with the speed-up techniques developed for road networks.

In this paper, we study the behavior of graph-based models in the prominent case of *dynamic* scenarios, i.e., when delays might occur to the original timetable. In particular, we make the following contributions. First, we consider the graph-based *reduced time-expanded model* and give a simplified and optimized routine for handling delays, and a re-engineered and fine-tuned query algorithm. Second, we propose a new graph-based model, namely the *dynamic timetable* model, natively tailored to efficiently incorporate dynamic updates, along with a query algorithm and a routine for handling delays. Third, we show how to adapt the ALT algorithm to such graph-based models. We have chosen this speed-up technique since it supports dynamic changes, and a careful implementation of it can significantly boost its performance. Finally, we provide an experimental study to assess the effectiveness of all proposed models and algorithms, and to compare them with the array-based state of the art solution for the dynamic case. We evaluate both new and existing approaches by implementing and testing them on real-world timetables subject to synthetic delays.

Our experimental results show that: (i) the dynamic timetable model is the best model for handling delays; (ii) graph-based models are competitive to array-based models with respect to query time in the dynamic case; (iii) the dynamic timetable model compares favorably with both the original and the reduced time-expanded model regarding space;

☆ Research partially supported by the Italian Ministry of University and Research under Grant 2012C4E3KT PRIN 2012 (AMANDA), as well as by EU FP7/2007–2013 under grant no. 288094 (eCOMPASS), no. 609026 (MOVESMART), and no. 621133 (HoPE). A preliminary version of this work appeared in [10].

* Corresponding author.

E-mail addresses: alessio.cionini@gmail.com (A. Cionini), gianlorenzo.dangelo@gssi.it (G. D'Angelo), mattia.demidio@gssi.infn.it (M. D'Emidio), daniele.frigioni@univaq.it (D. Frigioni), gianakok@ceid.upatras.gr (K. Giannakopoulou), paraskevop@ceid.upatras.gr (A. Paraskevopoulos), zaro@ceid.upatras.gr (C. Zaroliagis).

https://doi.org/10.1016/j.jda.2017.09.001

(iv) combining the graph-based models with speed-up techniques designed for road networks, such as ALT, is a very promising approach.

1. Introduction

Computing the *best route* in a *schedule-based public transportation system* (consisting, e.g., of trains, buses, trams, etc) is a problem that has been faced at least once by everybody who ever traveled. In particular, the so-called *journey planning* problem is defined as follows: given an input *timetable* associated to the transportation system, one wants to answer queries like: “What is the best route from some station A to some other station B if I want to depart at time t?”.

Despite its simple formulation, the journey planning problem is much more challenging with respect, for instance, to the classic route planning problem in road networks. In fact, schedule-based transportation systems exhibit an inherent time-dependent component that requires more complex modeling assumptions in order to obtain meaningful results. For this reason, nowadays, public transportation companies have invested a lot of effort in developing on-line software, called *journey planners*, which are able to efficiently answer to such kind of queries and to provide best routes with respect to some desired metrics.

Depending on the considered metrics and modeling assumptions, the journey planning problem can be specialized into a plethora of optimization problems. A first set of problems can be defined by considering different objective functions. If, for instance, the best route (a.k.a. *optimal*) is the one that minimizes the traveling time required to complete the journey, then the problem is called the *earliest arrival time problem*. Another, yet simpler and less considered, variant of the problem is to find the best route that minimizes the number of times that a passenger has to move from one vehicle of the system to another, during the journey. This version of the problem is called the *minimum number of transfers problem*. Sometimes, these two optimization criteria might be considered in combination (*bi-criteria problem*), possibly alongside further criteria (like, e.g., monetary cost). In this case, the problem is generally referred as the *multi-criteria problem*.

Further specializations can be obtained according to the level of abstraction at which the problem has to be solved. If, for instance, one wants to model (and optimize) the time required by a passenger for moving from one vehicle to another one within a station (i.e., *transfer time*), then the problem is referred to as *realistic* [31] while, when such an issue is ignored, the problem is referred to as *ideal* or *basic*. In this paper, we focus only on the realistic journey planning problem. Furthermore, if the departure time is within a time interval rather than be a single value, then the problem is called *profile* or *range* problem. We refer to the very recent survey of Bast et al. [3] for a comprehensive overview on journey planning.

1.1. Related work

The great variety of models proposed in the literature to solve the mentioned variants of the journey planning problem can be broadly classified into two categories: those representing the timetable as an *array*, and those representing it as a *graph* (see e.g., [3]).

Two of the most successful (and effective) examples of the array-based model are the Connection Scan Algorithm (CSA) [20] and the Round-bAsed Public Transit Optimized Router (RAPTOR) [17]. In CSA all the *elementary connections* of a timetable are stored in a single array which is scanned only once per query. The acyclic nature of some timetables is exploited to solve the earliest arrival problem. 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 bi-criteria problem. Recently, some faster approaches in terms of query time with respect to the two aforementioned methods, have been proposed in [18] and in [37].

The graph-based models, instead, store the timetable as a suitable graph and execute known adaptations of Dijkstra’s shortest path algorithm to compute optimal routes. There exist two main approaches of this kind: the *time-expanded* and the *time-dependent* model [31]. 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 with respect to the time-dependent model, and thus larger query times. A variant of the time-expanded model having a smaller number of nodes and arcs (the so called *reduced time-expanded model*) has been proposed in [31]. Graph-based models can in general be easily adapted to solve the multi-criteria problem as well as RAPTOR [3].

Experimental results have shown so far that the array-based approaches are faster in terms of query time than graph-based ones [3,19,20], as they are able to better exploit data locality and do not rely on priority queues. On the other hand, array-based approaches have been tested on metropolitan-size instances that span a time period of one day, while graph-based approaches can be used in periodic timetables (i.e., that span more than one day), and therefore the latter are more suitable to handle queries in large-scale networks. Moreover, during the last years, a great research effort has been devoted to devise many so-called *speed-up techniques*, which heuristically speed up Dijkstra’s algorithm for shortest paths (see, e.g., [3,4]). 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 (see, e.g., [5,16]). 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 [16].

Another limit of the graph-based models seems to be that they are not suitable to incorporate dynamic changes in the timetable. In fact, if the time duration of some connection changes (e.g., due to the delay of a train), the graph may not properly represent the modified timetable, and hence the computed route could be not optimal or even not feasible. As an example, a case study for the public transport system of Rome has shown that exploiting the published timetable does not lead to optimal or nearly-optimal routes [23]. 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) [15].

Moreover, although some speed-up techniques have been proposed for road networks which allow to handle dynamic updates [6,11–14,21,33,36], these techniques are in turn 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 is later exploited to answer queries. When a timetable modification occurs, the preprocessed information is 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 transportation system [1], has been studied in [2]. 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 [30].

Regarding array-based models, CSA and RAPTOR are suitable to handle dynamic changes of the timetable since they are not based on preprocessing, although there is no experimental evidence of this feature in the literature. Concerning the fast approaches in terms of query time given in [18] and in [37], the first is clearly not applicable to dynamic scenarios due to its huge preprocessing time, while the second looks more suitable, even though it needs around 6 minutes of (single threaded) preprocessing on a snapshot that models the public transportation system of London (with around 5 millions connections) in a fixed time interval.

1.2. Our contribution

This work aims at overcoming the limits of graph-based models by improving their performance, in order to be suitably used in realistic dynamic scenarios, while still offering competitive query times. More specifically, we make the following contributions.

- First, we focus on the realistic and reduced time-expanded models by providing a simplified and optimized version of the update routine in [15]. This new routine is used in combination with the dynamic packed-memory graph structure [29], which is able to efficiently handle topological changes to the graph. Furthermore, we heuristically improve the query algorithm for the time-expanded models, which results in significant improvements in query time.
- Second, we propose a new graph-based model for representing timetable information, called *dynamic timetable model* (DTM in short), that reduces the number of changes needed in the graph as a consequence of a timetable modification. Alongside we give both a query and an update algorithm for handling delays. The DTM model does not require any topological change to the graph and updates only few arc weights. At the same time, DTM is not based on time-dependent arc-weights, thus allowing to easily incorporate realistic constraints. Moreover, DTM produces a graph having a smaller number of nodes and arcs than the reduced time-expanded graph [31], and therefore, occupies less memory space. Also, this new model is implemented in combination with the dynamic graph structure in [29].
- Third, since both the above models are based on graph representations, 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 [25] to such models. We have chosen ALT since it supports dynamic changes [13], and since a careful implementation of it can significantly boost its performance [22].
- Fourth, we conducted a comparative experimental study of all these implementations on several local and long-distance European public transportation timetables, including the country-sized timetables of Sweden and Switzerland, and the metropolitan-sized timetables of Athens, Rome, Madrid, Berlin and London. Our study shows that: (i) Both the reduced time-expanded model and DTM require negligible update time (order of microseconds) after the occurrence of a delay. In particular, the time required by DTM is always the smallest one among all the tested models. (ii) The space required by DTM is smaller than that required by the time-expanded models. (iii) The new heuristic query algorithm for the reduced time-expanded model combined with ALT outperforms other existing methods and demonstrates query times similar to those of array-based methods. In particular, we compared the query time of our graph-based models against RAPTOR [17] on the largest metropolitan area instance at our disposal (London). Our experiments showed that graph-based models are competitive to RAPTOR in terms of query time, and that the reduced time-expanded query algorithm, when equipped with ALT, is faster than RAPTOR.

1.3. Overview of the paper

The paper is organized as follows. In Section 2 we give our notation and definitions for the considered problems. In Section 3 we describe the realistic time-expanded model, while in Section 4 we focus on its reduced version and provide an engineered query algorithm along with a simplified and optimized update routine. In Section 5 we introduce our new model along with its query and update algorithm and the corresponding proofs of correctness and complexity. In Section 6, we show how to adapt the unidirectional ALT algorithm to graph-based models. In Section 7 we present an experimental study to assess the performance of all presented approaches. Finally, in Section 8 we conclude the paper and suggest some future research directions.

2. Preliminaries

A *timetable* consists of data concerning: stations, vehicles (like, e.g. trains, bus or any other means of public transportation) 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 *vehicles*, $\mathcal{B}$ is a set of *stations* (sometimes in the literature also referred as *stops*), 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 vehicle $Z \in \mathcal{Z}$ leaves station $S_d \in \mathcal{B}$ at time t_d, and the immediately next stop of vehicle 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 $\{0, 1, \dots, 1439\}$ representing time in minutes after midnight. We assume that $|\mathcal{C}| \geq \max\{|\mathcal{B}|, |\mathcal{Z}|\}$, as we do not consider vehicles 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. $\Delta(t_1, t_2) = t_2 - t_1 (\bmod\ 1440)$. 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 follows that 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) is always smaller than 1440, for each $S \in \mathcal{B}$. An *itinerary* (a.k.a. a *journey*) in a timetable $\mathcal{T}$ is a sequence of elementary connections $P = (c_1, c_2, \dots, c_k)$ such that, for each $i = 2, 3, \dots, k$, $S_a(c_{i-1}) = S_d(c_i)$ and

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

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 plus the associated transfer times, i.e. $\Delta(P) = \sum_{i=1}^{k} \Delta(c_i) + \Delta(t_a(c_i), t_d(c_i + 1))$.

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 vehicle transfers. Such two criteria define the following optimization problems ([31]):

- 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 arrives at T as early as possible.
- 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 vehicle to another one as possible.

Given a timetable $\mathcal{T}$, a delay occurring on a connection c is modeled as an increase of d minutes on the arrival time, $t'_a(c) = t_a(c) + d(\bmod\ 1440)$. 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 vehicles that depend on $Z(c)$ in $\mathcal{T}$ (see e.g. [8,9,24,28,32] for examples of policies used to update a timetable).

We assume that the policy adopted is that no vehicle waits for a delayed one. Therefore, when a delay occurs on a connection c, 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 $d(\bmod\ 1440)$.

3. The realistic time-expanded model

In this section, we briefly summarize the characteristics of the realistic time-expanded model, along with its query and update algorithms.

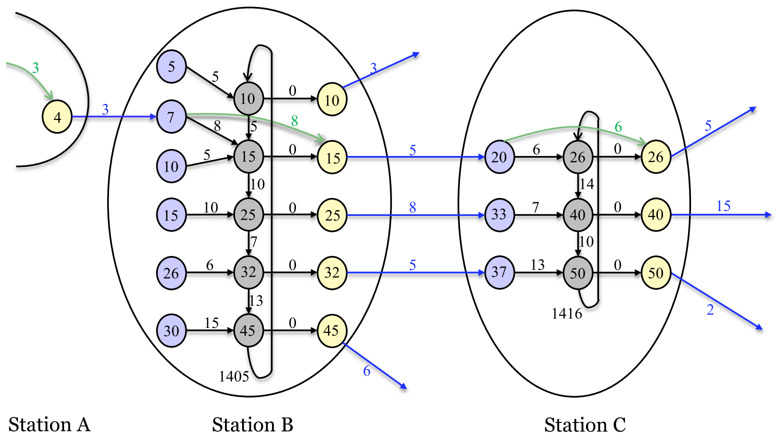

Fig. 1. A realistic time-expanded graph. Arrival, transfer, and departure nodes are drawn in blue, gray and yellow, respectively. Connection and arrival-departure arcs are drawn in blue and green, respectively. Arcs with at least one transfer node as endpoint are drawn in black. The minimum transfer time is 5 mins. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)

3.1. Model

In the realistic time-expanded model [31], 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 timestamp 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 w.r.t. timestamp transfer node v such that $\Delta(t(u), t(v)) \geq \mathit{transfer}(S)$.

To ensure the possibility to stay in the same vehicle 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 vehicle. Further, to allow transfers to an arbitrary vehicle, transfer nodes are ordered in a non-decreasing way with respect to the timestamps. Two adjacent nodes, with respect to the above ordering, are connected by a *transfer-transfer* arc from the smaller to the bigger, in terms of timestamp, node. To allow transfers over midnight, an *overnight-arc* from the node with the biggest timestamp to the one with the smallest timestamp is created. The weight of such arc is equal to the difference of the two timestamps modulo 1440. For each arc $e = (u, v)$, in the realistic 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. Fig. 1 shows a realistic time-expanded graph.

3.2. Query algorithm

In a realistic time-expanded graph G, given a timetable query (S, T, t_S), the earliest arrival problem can be solved by computing a shortest path from s to t in G, for instance by running Dijkstra's algorithm [3] and quitting the algorithm as soon as t is settled. In particular, s is the transfer node with the smallest timestamp within station S such that the timestamp associated to s, namely $t(s)$, is greater or equal than t_S. If no such node exists, then s is selected as the node among the transfer nodes of S such that $t(s)$ is minimum. Node t is an arrival node within station T with minimum distance to s, i.e., the first node of T that is extracted from Dijkstra's priority queue.

Note that the realistic time-expanded graph can easily 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 both cases, the query algorithm requires $O(|\mathcal{C}| \log |\mathcal{C}|)$ time in the worst case. In the remaining of the paper, for the sake of simplicity, we denote the above query algorithm as `TE-Q`.

3.3. Update algorithm

We now show how the realistic time-expanded graph has to be updated in order to reflect possible delays occurring on the input timetable, i.e., when a vehicle is delayed, by presenting the approach in [15]. In the remainder of the paper, for the sake of simplicity, we denote such update algorithm as `TE-U`. In particular, when a vehicle is delayed, it is not sufficient to simply increase the travel time of the delayed vehicle in the time-expanded model, i.e., the weight of the connection arc. Instead, also arcs within stations, like e.g., arrival-transfer arcs, have to be updated; that is, we have to change their weight and/or *rewire* them. As a consequence the update routine consists of three steps as follows.

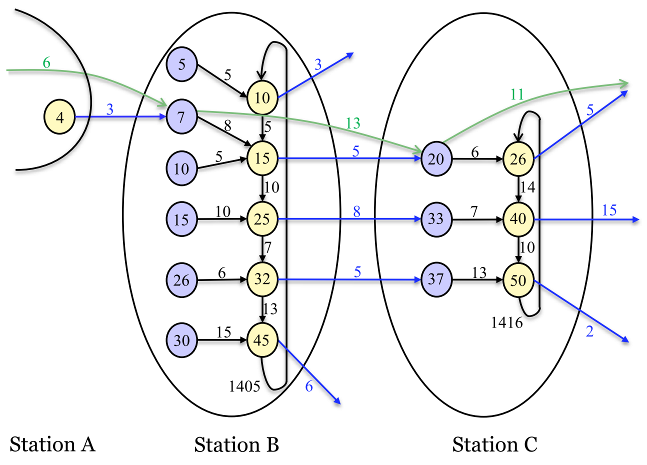

Fig. 2. The reduced time-expanded graph corresponding to the realistic time-expanded graph of Fig. 1. Arrival nodes are drawn in blue while departure nodes, ordered by departure time, are drawn in yellow. Arrival nodes are now connected directly to departure nodes. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)

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

Step 2 (Update Station Arcs). For all subsequent stations the delayed vehicle 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* the weight of each arrival-transfer arc by Δ.

Step 3 (Validate Station Arcs). The final step checks for every altered arrival-transfer arc, whether these arcs are still *valid*, i.e., the arc weight, after the increase of Δ, is still bigger than the transfer time of the associated arrival station. If the arc is still valid, we are done. Otherwise, the arc has to be *rewired*. That is, the target node has to be changed to the next "reachable" transfer node, namely the first node resulting in a valid arrival-transfer arc, i.e., the one with a timestamp greater than the updated arrival time of the vehicle.

Note that the first two steps require $O(1)$ and $O(|\mathcal{C}|)$ time, respectively. The third step, instead, requires $O(|\mathcal{C}|\log|\mathcal{C}|)$ because a binary search over the departure nodes is required in order to properly rewire each non-valid arc.

4. The reduced time-expanded model

In this section, we describe a variant of the realistic time-expanded model, originally introduced in [31], called *reduced* (realistic) time-expanded model, which we adopted in order to decrease the size of the realistic time-expanded graph. For this model, we provide a re-engineered and fine-tuned query algorithm, as well as a simplified and optimized update routine for handling delays.

4.1. Model

The main difference between the reduced and the original time-expanded model is that, in the reduced version, both transfer nodes and transfer-departure arcs are removed from the graph, without any loss of information and abstraction. In particular, departure nodes are merged with their corresponding transfer nodes and arrival nodes are connected directly to departure nodes. Also, if there is an arrival-departure arc between two stations s_1 and s_2 in the original realistic time-expanded graph, then such an arc is transformed to directly connect the corresponding arrival nodes of s_1 and s_2 in the reduced model.

This results in a reduction in the graph size by $|\mathcal{C}|$ nodes and $|\mathcal{C}|$ arcs, and therefore in a shorter traversal time of the graph. Fig. 2 shows the reduced time-expanded graph corresponding to the realistic time-expanded graph of Fig. 1.

4.2. Query algorithm

In what follows, we propose a variant of the Dijkstra's algorithm used for the realistic time-expanded graph, named TE-QH, which exploits the model structure and some restrictions, in order to improve the query time. Note that the asymptotic worst-case behavior is not affected, and that the query algorithm still requires $O(|\mathcal{C}|\log|\mathcal{C}|)$ time. However, in Section 7 we will show that our approach results in a much better query time in practice. More specifically, our approach is as follows.

First, 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 highly depends on the number of priority queue

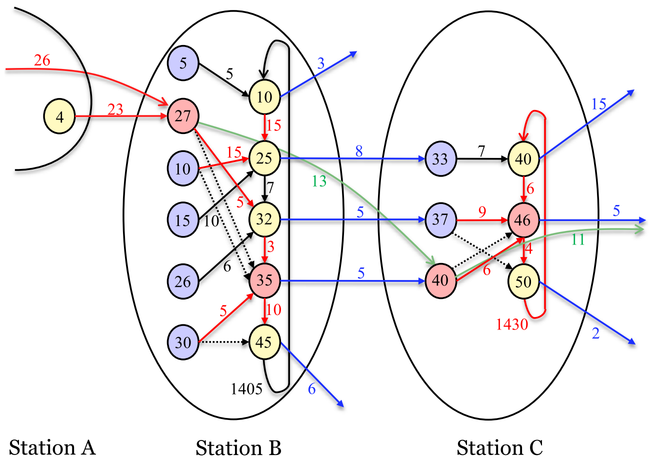

Fig. 3. Handling delays in the reduced time-expanded model. Those arcs, as well as departure and arrival nodes of the delayed vehicle that need to be updated are drawn in red. The delay is 20 mins. Dotted arcs are those existed before the delay and which after the update procedure are removed. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)

operations, whose computational cost depends, in turn, on the overall size of the queue. However, in order to preserve the correctness of the algorithm with respect to the original query algorithm, we extend the arc relaxation depth only for the case of departure nodes. That is, for each arc relaxation, when the head of an arc is a departure node and its distance label is updated, we also forward the arc relaxation to the outgoing arcs of that node. In that case, the arc relaxation depth is upper bounded by the number of departure nodes, exploiting a suitably defined acyclic component between the adjacent stations. This acyclic component, between two adjacent stations s_A and s_B, consists of all $v_1, v_2, \ldots, v_{k-1}, v_k$ simple paths, where v_1 is an arrival node of s_A, $v_2, .., v_{k-1}$ departure nodes of s_A, and v_k is an arrival node of the neighbor station s_B. For example, in Fig. 2, the acyclic subgraph, between stations B and C, is induced by arrival node 7 and departure nodes 15, 25, 32 in station B and arrival nodes 20, 33, 37 in station C, and also via the corresponding 7–15–20, 7–15–25–33 and 7–15–25–32–37 paths.

Second, in order to narrow the search space, we modify the node exploration criteria in order to be more selective. In particular, taking as a reference the source station s_S, while executing Dijkstra's algorithm, we keep trace of the earliest arrival time to stations that are currently reached 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 goal is to exclude from the search those that are not part of an optimal path. To achieve this, we exploit the fact that departure nodes are sorted in non-decreasing order with respect to the associated timestamp. Therefore, at a certain step of the execution of the query algorithm, we can skip (i.e., exclude from the search) all those departure nodes whose departure time is higher than the current 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 a solution to the earliest arrival problem for the query. Note that the proposed heuristic can be clearly used in combination with other speed-up techniques (such as ALT). We will show in the experimental section that such combination provides a significant improvement in the performance of the query algorithm.

4.3. Update algorithm

In what follows, we describe how we have engineered, simplified, and optimized the update routine of Section 3.3, designed for realistic time-expanded graphs, in order to work also on reduced time-expanded graphs. Note that, the worst-case asymptotic time is not affected, but the new update algorithm is significantly faster in practice. In the remaining of the paper, for the sake of simplicity, we denote the following update algorithm as TE-UH.

For a better illustration of TE-UH, we use throughout its description a running example based on Station B of Fig. 2 (before a delay) and Fig. 3 (after a delay). Let the vehicle's delay be $\Delta = 20$ mins. Firstly, we increase by Δ the weight of the incoming arcs of the arrival node in the station (B) which correspond to the delayed vehicle (the incoming arcs of arrival node 7 in Fig. 2, and the incoming arcs of delayed arrival node 27 in Fig. 3). Secondly, for each station that the delayed vehicle passes through, we update the affected arrival-departure and departure-departure arcs, by changing their weight and/or rewiring them. In each such station, a pair of a delayed arrival node x (the arrival node 7 in Fig. 2, and the delayed arrival node 27 in Fig. 3) and a delayed departure node y (the departure node 15 in Fig. 2, and the delayed departure node 35 in Fig. 3) is introduced. 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 of the topology of the reduced time-expanded graph, in order to remain valid. The affected arcs are only within stations, and specifically those satisfying one of the following conditions:

- Their tail endpoint is the delayed arrival node x (the arc from arrival node 7 to departure node 15 in Fig. 2, and the arc from arrival node 27 to departure node 32 in Fig. 3).
- Their head endpoint is the delayed departure node y, through which the vehicle continues its travel to another station (the arc from arrival node 10 to departure node 15 in Fig. 2, and the arc from arrival node 10 to departure node 25 in Fig. 3).
- Their head endpoint is the successor (in time) of departure node of y (the arc from arrival node 30 to departure node 45 in Fig. 2, and the arc from arrival node 30 to departure node 35 in Fig. 3).

To maintain the invariant of keeping the departure nodes sorted according to their timestamp, we have to move the delayed departure node in its proper position. This is performed in a way similar to moving a node from one location to another in a linked list. Then, we only need to link the affected arrival nodes with the proper departure nodes so that transfer times constraints within the station are still satisfied. Alongside, we update the arcs with the new correct weights. The above operations require 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 arcs constant. Fig. 3 shows the result of execution of the presented update algorithm on the reduced time-expanded graph of Fig. 2, in case of a 20 minutes delay.

The only disadvantage of this approach 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 that we conducted showed that the new (simplified and optimized) routine is at least 50% faster than the original one.

5. The dynamic timetable model

In this section, we introduce our new graph-based approach, called *dynamic timetable model* (DTM for short), along with a suitable query algorithm, to solve both EAP and MNTP, and an update algorithm for handling delays.

5.1. Model

Given a timetable $\mathcal{T} = (\mathcal{Z}, \mathcal{B}, \mathcal{C})$, we define a directed graph $G = (V, E)$ called *dynamic timetable graph* and an associated 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}), called *connection arc* of c, 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 vehicle $Z \in \mathcal{Z}$ which travels through the itinerary $(c_1, c_2, \ldots, c_k)$, an arc, called *vehicle 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 $i = 1, 2, \ldots, k-1$;
- The weight of each connection arc (d_c, s_{S_a}) is set to $w(d_c, s_{S_a}) = \Delta(t_a(c), t_d(c))$;
- The weight of each vehicle arc $(d_{c_i}, d_{c_{i+1}})$ is set to $w(d_{c_i}, d_{c_{i+1}}) = \Delta(t_d(c_i), t_d(c_{i+1}))$;
- The weight of each switch arc is set to a default infinity value.

In each station, the departure nodes d_c are ordered by their arrival time at the immediately next station S_a, i.e., by the value $t_d(c) + w(d_c, s_{S_a})$. Moreover, for each switch node s_S, we store the station S it is associated with while, for each departure node d_c, we maintain both the departure time reference $t_d(c)$ and the vehicle $Z(c)$ of connection c which d_c is associated with. Fig. 4 shows the dynamic timetable graph corresponding to the realistic time-expanded graph of Fig. 1.

Theorem 1. *Storing the dynamic timetable graph G associated to a timetable $\mathcal{T} = (\mathcal{Z}, \mathcal{B}, \mathcal{C})$ requires $O(|\mathcal{C}|)$ space.*

Proof of Theorem 1. By the given construction, a dynamic timetable graph G has $|\mathcal{B}| + |\mathcal{C}|$ nodes and a number of arcs which is smaller than or equal to $3|\mathcal{C}|$. 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, respectively. The theorem now follows, since $|\mathcal{C}| \geq \max\{|\mathcal{B}|, |\mathcal{Z}|\}$. □

5.2. Query algorithm

In what follows, we present the query algorithm, named `DTM-Q`, suitable for solving both EAP and MNTP on DTM graphs. We first describe the case of EAP queries and then show how to adapt the algorithm to answer MNTP queries. Given a *dynamic timetable graph* G, an EAP query (S, T, t_S) can be answered by executing a modified Dijkstra's algorithm on G, starting from the switch node s_S of S, that will be described below.

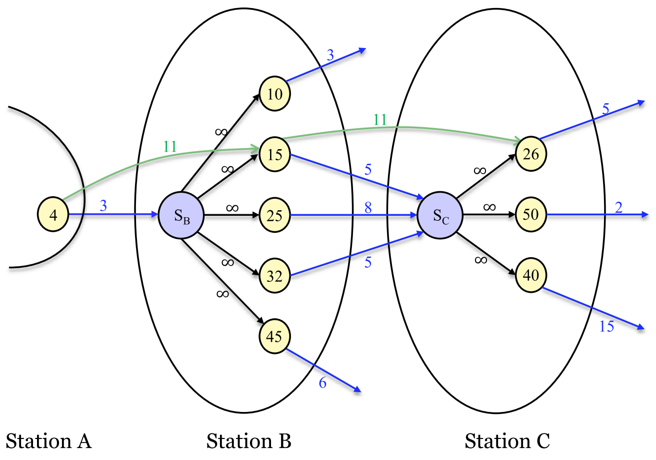

Fig. 4. The dynamic timetable graph corresponding to the realistic time-expanded graph of Figure 1. Switch nodes are drawn in blue while departure nodes, ordered by arrival time, are drawn in yellow. Inside each departure node the departure time of the corresponding elementary connection is reported. Connection arcs are drawn in blue, switch arcs are drawn in black, while vehicle arcs are drawn in green. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)

Before discussing the details of the algorithm, we first describe the additional data structures used by DTM-Q with respect to the classic Dijkstra's algorithm. In particular, for each switch node s_A of a station A, algorithm DTM-Q stores:

- A vector of boolean flags D_A, whose size is given by the number of stations A' such that there exists an elementary connection departing from A and arriving at A'. We denote the element of D_A associated to A' as $D_A[A']$. Initially, all flags of D_A are set to false.
- Along with the distance from s_S to s_A in the priority queue used by Dijkstra's algorithm, the associated connection c' (is stored) such that the arc $(d_{c'}, s_A)$ is the one through which the particular distance has been obtained during a relaxation step. For non-switch nodes such an associated connection is assumed to be equal to a default value NIL.

Given the EAP query (S, T, t_S), the DTM-Q algorithm works as follows. First, it starts by inserting the switch node s_S of the departure station S in the priority queue. The distance and connection associated with node s_S in the queue are set to 0 and NIL, respectively. Then, the visit proceeds by extracting nodes, one by one, from the priority queue in a Dijkstra-like fashion, while the queue is not empty. When a node is extracted, the behavior of the algorithm depends on the type of the extracted node.

Departure nodes are processed like in the classic Dijkstra's algorithm, i.e., (i) outgoing arcs are scanned and relaxed in order to discover shorter paths (if any); (ii) associated neighboring (either departure or switch) nodes are, according to the weights of the arcs, either inserted in the priority queue, if they were not present, or their distance is decreased, if possible, otherwise.

Switch nodes are processed as follows. Let us consider the time when a switch node s_A, associated with a station $A \in \mathcal{B}$, is extracted from the priority queue. Let $dist(s_S, s_A)$ be the distance from s_S to s_A associated to s_A in the queue and let c' be the associated elementary connection through which $dist(s_S, s_A)$ has been obtained. The value of $dist(s_S, s_A)$ essentially corresponds to the minimum time required to reach station A from station S, departing at time t_S. Hence, the algorithm first computes the value $x = t_d(c') + w(d_{c'}, s_A) \pmod{1440}$, which represents the arrival time of connection c'. If c' is equal to NIL, then the algorithm assigns $x = t_S$. Then, for each outgoing switch arc (s_A, d_c) such that $S_d(c) = A$ and $t_d(c) \geq x$, the algorithm performs a so-called *enabling* phase, that is, it *enables* arc (s_A, d_c) if $D_S[S_a(c)] = \text{false}$ and

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

The enabling operation on arc (s_A, d_c) consists in:

- setting the weight $w(s_A, d_c)$ to $\Delta(x, t_d(c))$;
- either inserting d_c in the priority queue, if it was not present, or in decreasing its distance, if possible, otherwise.

This enabling phase clearly stops (and hence the Dijkstra's search is pruned) when the vector D_A contains only true values, which are set in the following case.

Let (s_A, d_c) be the switch arc with the smallest arrival time that is enabled for some station $S' = S_a(c)$. Then, the algorithm sets $D_A[S']$ to *true* when an arc $(s_A, d_{c'})$ such that $S_a(c') = S'$ and $\Delta(t_a(c), t_a(c'))\pmod{1440} > \mathit{transfer}(S')$ is scanned (the time instances $t_a(c)$ and $t_a(c')$ required to check such a condition can be computed by using x, $t_d(c)$, $t_d(c')$

and the associated arc weights). Since departure nodes are ordered by arrival time, this guarantees that the algorithm ignores any connection (in the form of departure nodes) leaving A towards S' after this step, as it cannot lead to a better solution to the considered EAP query.

In other words, 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 by a value greater than *transfer*(B), then only the one with smallest arrival time is considered (i.e., enabled) while the other one is essentially discarded, since it will not lead to a better solution to reach B. This is obtained by suitably setting the weights and the values of D_A. In fact, if we assume that $x \le \min\{t_d(c_1), t_d(c_2)\}$ and $t_a(c_1) < t_a(c_2) + \mathit{transfer}(B) (\bmod\ 1440)$, i.e., 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 $\mathtt{true}$ when arc (s_A, d_{c_2}) is scanned, and weights are set to $w(s_A, d_{c_1}) = t_d(c_1) - x$ and $w(s_A, d_{c_2}) = \infty$. Hence, d_{c_2} will never be part of a solution to the associated EAP query. Note that ties can be broken arbitrarily.

The overall DTM-Q search is stopped as soon as the switch node s_T, associated to the arrival station T, is extracted from the queue. The arrival time t_T, which is given by $\mathit{dist}(s_S, s_T) + t_S$, is accordingly returned.

In order to clarify the behavior of DTM-Q, we show an example of its execution on the graph of Fig. 4 with EAP query $(S, T, t_S) = (B, C, 13)$. We assume $\mathit{transfer}(A) = \mathit{transfer}(B) = \mathit{transfer}(C) = 5$ mins. The algorithm starts by enqueueing s_B with distance equal to 0 and default associated connection. Moreover, *transfer*(B) is set to 0. Then, while the queue is not empty, it proceeds by extracting nodes one by one. When s_B is extracted with distance $d = 0$, its outgoing arcs are scanned, starting from the one leading to a departure node d_i whose departure time t_{d_i} is larger than or equal to $x = t_S = 13$. In our example, the first node of this kind is the one labeled with 15 in Fig. 4. Let us denote this node by b_1, with $t_{b_1} = 15$, and let us assume that associated connection is c_1 (arc $(15, s_C)$). At this point, the enabling phase starts. In particular, we first enable the switch arc leading to b_1 by setting its weight to $\Delta(x, t_{b_1}) = 15 - 13 = 2 \ge \mathit{transfer}(B)$ and by inserting b_1 into the queue, with distance 2. Then, we scan the next arc, the one leading to the node labeled with 25 in Fig. 4. Let us denote this node by b_2, with $t_{b_2} = 25$, and let us assume that its associated connection is c_2 (arc $(25, s_C)$). Note that this connection is linking a departure node of station B again with the switch node of station C. The switch arc (s_B, b_2) has weight $\Delta(x, t_{b_2}) = 25 - 13 = 12 \ge \mathit{transfer}(B)$. Hence, we check whether we are meeting an arc (s_B, b_2) such that $S_a(c_2) = C$ and $\Delta(t_a(c_1), t_a(c_2))(\bmod\ 1440) > \mathit{transfer}(C)$, where $t_a(c_1) = t_{b_1} + w(c_1) = 15 + 5 = 20$ while $t_a(c_2) = t_{b_2} + w(c_2) = 25 + 8 = 33$ and $\Delta(t_a(c_1), t_a(c_2))(\bmod\ 1440) = 13$ which is clearly greater than $\mathit{transfer}(C) = 5$. Therefore, we have found a connection that leads to the same station C, fulfills Inequality (1), and has two arrival times that differ for a value greater than *transfer*(C). Hence, we set $D_B[C]$ to true and prune the search for all following connections leading to C since none of them can induce a better solution to reach C, i.e. we skip any further switch arc (s_B, b_i) and its associated connection c_i, such that $S_a(c_i) = C$. The search continues by extracting b_1 and by relaxing its outgoing arcs, thus inserting s_C into the queue with distance $\mathit{dist}(s_B, b_1) + w(c_1) = 2 + 5 = 7$, along with the departure node labeled with 26 in Fig. 4, because of the outgoing vehicle arc of b_1, say v_1, with distance $\mathit{dist}(s_B, b_1) + w(v_1) = 2 + 11 = 13$. The next step of the algorithm consists in extracting s_C which triggers the termination criterion of DTM-Q, since the switch node s_C, associated to the arrival station C, is extracted from the queue. Thus, the algorithm returns an itinerary of total (minimum) length equal to 7, which induce an according arrival time $t_C = 13 + 7 = 20$.

Algorithm DTM-Q can be adapted to answer a MNTP query (S, T, t_S), by slightly modifying it as follows. First, vector D is not needed. Second, when a switch node s_A is extracted from the priority queue with associated connection c', then all the switch arcs outgoing it that satisfy transfer time constraints (Inequality (1)) are enabled, and the weight of each switch arc (s_A, d_c) is set to 0, if $Z(c) = Z(c')$, and to 1 otherwise.

Theorem 2. *DTM-Q solves EAP and MNTP in $O(|\mathcal{C}| \log |\mathcal{C}|)$ worst case time.*

Proof of Theorem 2. Given an EAP query (S, T, t_S), by the construction of the dynamic timetable graph, the path found by DTM-Q is an itinerary in $\mathcal{T}$ starting from S at time $t \ge t_S$ 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 Dijkstra's algorithm. In particular we prove that, when a switch node s_A is extracted from the Dijkstra's queue, the value of $\mathit{dist}(s_S, s_A)$ is minimum, therefore inducing 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, extracted from the queue, whose value of $\mathit{dist}(s_S, s_A)$ is not minimum. Clearly $S_A \ne S_S$. Moreover, 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 $(d(c'), s_A)$ and a switch arc $(s_B, t_d(c'))$, for some station B and connection c'. Furthermore, during the enabling phase, the switch arc $(s_B, t_d(c'))$ is enabled (i.e. its weight is set to a non-infinity value) only if, among all possible connections that connect station B to station A and that satisfy Inequality (1), c' is the one having minimum length $\Delta(c')$. Moreover, since the priority of s_A has been updated for the last time because of the relaxation of arc $(d(c'), s_A)$, then all the other connection arcs leading to s_A correspond to longer paths. Therefore, it must be the case that, when the switch node s_B of station B is extracted from the queue, $\mathit{dist}(s_S, s_B)$ is not minimum, which contradicts the fact that s_A was the first node extracted with non-minimum distance.

The computational complexity of DTM-Q is that of the Dijkstra's algorithm in a graph with $n = |\mathcal{B}| + |\mathcal{C}|$ nodes and $m \le 3|\mathcal{C}|$ arcs, plus the additional cost of the enabling step, which must be done, in the worst case, for each outgoing arc of a switch node, say s_A, extracted from the priority queue, 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 in DTM-Q while it requires

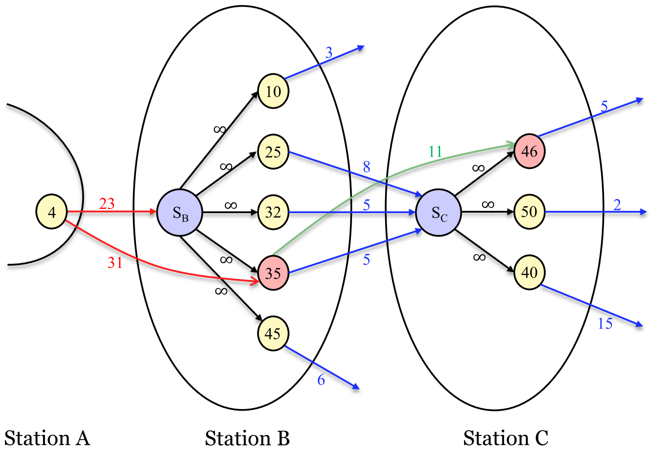

Fig. 5. Handling delays in the DTM model. A delay of 20 minutes induces two arc weight changes (red arcs) and the update of the time associated to the corresponding departure nodes (red nodes). (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)

constant time in Dijkstra's algorithm. Since D_A has at most $|\mathcal{B}|$ elements, this takes 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 of DTM-Q is $O(n \log n + m \log|\mathcal{B}|) = O(|\mathcal{C}| \log|\mathcal{C}|)$, as $|\mathcal{B}| \leq |\mathcal{C}|$.

The correctness and the complexity of the DTM-Q algorithm modified to answer to MNTP queries follow straightforwardly form the discussion above. □

5.3. Update algorithm

In what follows, we present the update algorithm, named DTM-U, for handling delays occurring on the given input timetable, and for reflecting such delays to the corresponding DTM graph.

Let us assume that we are given a timetable $\mathcal{T}$, a delay Δ occurring on a connection c of $\mathcal{T}$, and the corresponding *disposition timetable* $\mathcal{T}'$. If $\mathcal{T}$ is represented as a dynamic timetable graph G, then the DTM update algorithm computes the dynamic timetable graph G' corresponding to $\mathcal{T}'$ as follows. For the sake of clarity and in order to better illustrate how DTM-U works, similarly to what we did in Section 4.3, we make use of a running example based on the very same Station B of Fig. 4 (before a delay) and Fig. 5 (after a delay). We again assume that the vehicle's delay is $\Delta = 20$ mins.

First, the weight of both the connection arc $(d_c, s_{S'})$ and that of the vehicle arc $(d_c, d_{\hat{c}})$ is updated by adding the amount $\Delta(\text{mod } 1440)$, where $S' = S_a(c)$ and $d_{\hat{c}}$ is the departure node of S' such that $Z(c) = Z(\hat{c})$ (if any). An example of this behavior is shown in Fig. 5, where the weight of connection arc connecting departure node 4 of Station A to switch node s_B of Station B, and the weight of the vehicle arc connecting the same departure node to the departure node 15 of Station B, are increased by $\Delta = 20$.

Second, for each connection c' that is *affected* by the delay occurring on c, i.e. such that the departure time in $\mathcal{T}$ differs from that in $\mathcal{T}'$, the time reference of the departure node $d_{c'}$ is updated by adding $\Delta(\text{mod } 1440)$. An example of this behavior is shown in Fig. 5 where the departure nodes of Station B and C, originally having departure times 15 and 26 (see Fig. 4), respectively, are updated to the delayed departure nodes of Station B and C, having departure times 35 and 46, respectively.

Note that the search for affected connections c' can be done by performing a graph visit of G, starting from the departure node d_c, and by selecting, during the visit, all connections c' such that $Z(c') = Z(c)$. Note also that, some further computation is required in the case G is used to answer EAP queries. In this scenario, in fact, the update of arc weights and time reference might break the ordering within the array representing the arcs. In order to restore this ordering, it is enough to compare the new values of arrival times of the few changed arcs and swap them, within the array, if needed. For instance, departure node 15 of Station B in Fig. 4 is moved down by two positions in Fig. 5 in order to restore the proper ordering with respect to departure time, while departure node 26 of Station C in Fig. 4 keeps its position despite the increase.

Theorem 3. *DTM-U requires $O(|\mathcal{C}|)$ worst case time.*

Proof of Theorem 3. The first part of the proof aims at proving that G' is the dynamic timetable graph representing $\mathcal{T}'$.

First of all, note that no topological changes are performed on G. Therefore, in order to prove the claim, we simply point out that: (i) all arcs in G' associated with affected connections c' (i.e., of either connection or vehicle type) are properly weighted, according to the definition of dynamic timetable graph; in fact, for each of the delayed connection c, the weight of both the connection arc (d_c, s_A) and the vehicle arc $(d_c, d_{\hat{c}})$ (if any) are updated by adding the amount $\Delta(\text{mod } 1440)$; (ii) departure nodes have appropriate time reference; in fact, notice that the time reference of departure nodes $d_{c'}$ of affected connections c' is increased by the same amount $\Delta(\text{mod } 1440)$; (iii) in the case the dynamic timetable graph is used to solve EAP, departure nodes are ordered with respect to new arrival times; in this case, it is easy to see that the ordering can be restored, within the array, by moving the changed arcs in the proper position, according to the new updated arrival time.

The second part of the proof aims at proving that DTM-U computes G' in $O(|\mathcal{C}|)$ worst case time.

Let us denote by m_i the number of outgoing arcs from the switch node s_i of the DTM graph G, for each $i \in \mathcal{B}$. Then, if G is being exploited for answering MNTP queries, the algorithm, i.e., the search for affected connections c' and the corresponding time and weight updates, simply requires $O(\sum_{i\in\mathcal{B}} m_i) = O(m) = O(|\mathcal{C}|)$. Otherwise, if G is being exploited to answer EAP queries, then we have to take into account also the cost of reordering the array. This can be done in $O(\sum_{i\in\mathcal{B}} m_i) = O(m) = O(|\mathcal{C}|)$ by using Radix–Sort. Note that, since only one element is out of order in each array, the same result can be obtained by moving the unique entry whose time reference has changed to the right place in the array. This takes again $O(\sum_{i\in\mathcal{B}} m_i) = O(m) = O(|\mathcal{C}|)$ time. $\square$

Notice that Theorem 3 gives 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 the robust design of timetables [7–9,24,28,32]. We will show in Section 7 that the number of arcs that have to be updated as a consequence of a delay is always a small constant with respect to the graph size.

5.4. Engineering the query algorithm

Also for DTM, we propose an engineered variant of the basic query algorithm, named DTM-QH, which exploits the model structure and some restrictions to reduce the query time. In particular, we adopt the following approaches.

First, we propose and apply to DTM-Q a modification of the general node blocking technique (see [16]). The purpose of this strategy is that of pruning the search by reducing the size of the Dijkstra's priority queue. This is achieved by exploiting the structure of the timetable graph which allows to avoid the insertion of sets of nodes that are clearly not part of shortest paths. In particular, when performing the query algorithm, if nodes belonging to a station that has already been reached, i.e., such that its switch node has already been settled by the visit, are touched (either visited for the first time or not), they are immediately discarded and not inserted in the priority queue, thus inducing smaller overall queue size and reduced number of queue related operations (i.e., insertion, decrease-key and delete-min operations). The correctness in this case is easily guaranteed. In fact, if the switch node of a station has been settled, we can be sure, by the sub-optimality property of shortest paths, that no path shorter than that already discovered can be found. Note that this approach is somehow similar to the modified node exploration criterion we applied to the reduced time-expanded graph (see Section 4.2). However, in the case of DTM-Q, we do not need to store current arrival times for reached station, as switch nodes represent all arrival events of a station.

Second, we propose and apply to DTM-Q a "runtime" modification of the omitting nodes technique proposed for reducing the size of time-expanded graph in [31]. The original technique [31] aimed and allowed to (permanently) reduce the size of time-expanded graphs by short-cutting nodes (i.e., bypassing with two suitable arcs) with out-degree equal to one. In this case, we exploit the case of nodes with out-degree equal to one at runtime as follows: when the query routine touches a node x with out-degree equal to one, the relaxation depth is extended to two, node x is not inserted in the queue, and the target node of the single arc outgoing x is evaluated, to check whether it provides an improvement w.r.t. to the current shortest path search. It is easy to see that this strategy does not affect the correctness of the query algorithm, though it provides a clear reduction in the average queue size and in the number of queue related operations (i.e., insertion, decrease-key and delete-min operations).

Third, we propose and apply to DTM-Q a slightly different topological approach in order to reduce the unnecessary exploration and labeling on nodes and arcs during the query phase. In this variant, at each station, the switch arcs are grouped by adjacent arrival station and then for each group its arcs are sorted by arrival time. Therefore, when an adjacent arrival station S is reached, the whole group of the switch arcs heading to S can be skipped directly. Also in the procedure for handling delays this reduces the swap arc operations which are now limited only in the affected station's grouped arcs.

Note that these approaches are also independent and compatible with the use of other speed-up techniques (such as ALT).

5.5. Comparison with time-expanded models

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

In case of delays the DTM-U algorithm requires the same worst-case computational time with that of the update algorithms for the two time expanded models. However, 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., [15]). This behavior could imply a large computational time which depends on the way the graph is stored. On the contrary, DTM-U 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.

Although DTM 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 DTM requires $|\mathcal{B}| + |\mathcal{C}|$ nodes and at most $3|\mathcal{C}|$ arcs (we recall that $|\mathcal{C}| \geq |\mathcal{B}|$).

Even if the three query algorithms require the same worst-case computational time, the `DTM-Q` algorithm must perform the additional step of enabling arcs and computing the weights of the switch arcs.

6. Combining graph-based models with ALT

One of the advantages of graph-based models for timetable information systems, over the array-based ones, is that the former can exploit the so-called speed-up techniques, that have been developed during the last years for accelerating the computation of shortest paths in other important applications, like e.g., route planning in road networks. Indeed, many of such techniques can be adapted to be used in combination with graph-based models for timetable information systems and thus improve the query time.

In this context and in order to boost the performance of our query algorithms, we combined them with ALT [25], one of the simplest and most used speed-up techniques. ALT is a goal-directed technique, i.e., its main aim is that of pushing the shortest-path search faster towards the target node/station. Many experimental studies have shown that ALT provides very good speed-ups, notwithstanding its simplicity, and that the magnitude of the speed-up, in general, depends on the structure of the input graph.

The adaptation of ALT to the query algorithms of the time-expanded models is relatively easy, and several variants have already been proposed; see e.g., [16]. Hence, we shall not provide any details of this combination.

The DTM model is somehow different and thus we provide below the adaption of ALT to its query algorithm.

As already mentioned, the basic idea behind the goal-directed ALT algorithm is to direct the Dijkstra's search by pruning the corresponding search space towards the target t of the query. This is achieved by adding a *feasible potential* to the priority of each node in the queue. 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, it 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 [25] for more details). It is easy to see that the tighter the lower bounds are, the more narrowed the search space is (i.e., the faster the query algorithm performs). Therefore, choosing good landmarks that provide tight lower bounds is a fundamental part of the preprocessing phase of ALT. Note that many approaches have been proposed for the selection of landmarks in the literature.

For DTM, we apply an approach similar to that proposed in [16]. In particular, 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 among connections traveling from station A to 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 $O(|\mathcal{B}|^2)$ and by computing all-pairs shortest paths on the condensed version of the input graph (see [16]). This makes sense particularly when the stations are relatively few in number.

Finally, we combined ALT along with the heuristics described in Section 4.2 for reduced time-expanded graphs, and with the heuristic described in Section 5.4 for DTM graphs. We denote the two resulting algorithms as `TE-QH-ALT` and `DTM-QH-ALT`, respectively. The combination of the approaches reduces much more the search space size and leads to a more efficient algorithm.

7. Experimental analysis

In this section, we present an extensive experimental study to assess the performance of all algorithms presented in the previous sections. Our experiments have been performed on a workstation equipped with an Intel Quad-core i5-2500K 3.30 GHz CPU and 12 GB of main memory. All our implementations were done in `C++` (gcc compiler v4.6.3 with optimization level O3).

For all the considered graph-based models, we implemented: (i) the routines for building the graph; (ii) the basic query algorithms as well as their corresponding engineered heuristic versions for the reduced time-expanded model and the dynamic timetable model; (iii) the update routines for handling delays; (iv) the query algorithms empowered with the ALT technique. All implementations are used in combination with the dynamic packed-memory graph structure in [29] which is able to efficiently handle topological changes to the graph.

7.1. Input data and parameters

As input data to our experiments we used: (a) Three train timetables (namely efz, ger, and eur) and three bus timetables (namely bts, ks, and bvb) extracted from a large data set, provided by HaCon [26] for scientific use. (b) Eight different

Table 1
Tested timetables and sizes of the corresponding realistic time-expanded (*real*) and reduced time-expanded (*red*) graphs.

Source	$\mathcal{T}$	$\lvert\mathcal{B}\rvert$	$\lvert\mathcal{C}\rvert$	TE (real)		TE (red)	
				$\lvert V\rvert$	$\lvert E\rvert$	$\lvert V\rvert$	$\lvert E\rvert$
HaCon	bts	716	12 689	38 067	63 445	25 378	49 862
	ks	1 865	44 744	134 232	223 720	89 488	175 536
	efz	2 198	41 613	124 839	208 065	83 226	159 290
	bvb	2 874	292 542	877 626	1 462 710	585 084	1 154 792
	ger	6 493	428 982	1 286 946	2 144 910	857 964	1 668 171
	eur	7 786	596 129	1 788 387	2 912 210	1 192 258	2 316 081
GTFS	Victoria	2 230	403 709	1 211 127	2 018 545	807 418	1 604 359
	Madrid	4 636	2 070 490	6 211 470	10 352 450	4 140 980	8 200 875
	Athens	7 897	1 694 069	5 082 207	8 436 181	3 388 138	6 742 112
	Rome	8 560	2 727 672	8 183 016	13 550 426	5 453 344	10 822 754
	Berlin	12 838	4 322 549	12 967 647	21 612 745	8 645 098	17 024 138
	London	20 843	14 064 967	42 194 901	70 324 835	28 129 934	55 758 468
	Switzerland	31 213	7 097 772	21 293 316	33 961 786	14 195 544	26 841 306
	Sweden	50 387	5 884 019	17 652 057	29 169 177	11 768 038	23 285 158

Table 2
Tested timetables and sizes of the corresponding dynamic timetable graphs.

Source	$\mathcal{T}$	$\lvert\mathcal{B}\rvert$	$\lvert\mathcal{C}\rvert$	DTM	
				$\lvert V\rvert$	$\lvert E\rvert$
HaCon	bts	716	12 689	13 405	38 067
	ks	1 865	44 744	46 609	134 232
	efz	2 198	41 613	43 811	124 839
	bvb	2 874	292 542	295 416	877 626
	ger	6 493	428 982	435 475	1 286 946
	eur	7 786	596 129	603 915	1 719 952
GTFS	Victoria	2 230	403 709	405 939	1 200 658
	Madrid	4 636	2 070 490	2 075 124	6 130 385
	Athens	7 897	1 694 069	1 701 966	5 048 136
	Rome	8 560	2 727 672	2 736 232	8 097 361
	Berlin	12 838	4 322 549	4 335 387	12 701 695
	London	20 843	14 064 967	14 085 810	41 837 355
	Switzerland	31 213	7 097 772	7 128 985	20 521 440
	Sweden	50 387	5 884 019	5 934 406	17 402 734

timetable data sets in the General Transit Feed Specification (GTFS) format, containing various means of public transport. In particular, these data sets concern the metropolitan areas of Athens, Rome, Victoria, Madrid, Berlin, and London, and the country-sized timetables of Sweden and Switzerland. The London data set was obtained from [35], while the rest from [34].

For each timetable $\mathcal{T}$, we built: (i) the realistic time-expanded graph; (ii) the reduced time-expanded graph; and (iii) the dynamic timetable graph. For storing the graphs, we used the packed-memory graph data structure [29]. For the dynamic timetable graph, the outgoing switch arcs (s_S, d_c) of each switch node 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. We used a classic binary heap whenever a priority queue was needed.

Tables 1, 2 and 3 provide detailed information about the input timetables. In Tables 1 and 2, we report, for each timetable, the number of stations $|\mathcal{B}|$ and the number of elementary connections $|\mathcal{C}|$ between stations (a proxy for size), as well as the number of nodes $|V|$ and arcs $|E|$ of the corresponding graph, for each model. In Table 3, we report, for each timetable, the average transfer time for changing vehicles, the average number of adjacent stops or stations, and the percentage of the existed transportation means to the total number of the elementary connections. In the timetables of Madrid, Athens and Rome, the transfer times were not available, and we set it arbitrarily to 3 minutes.

Tables 1 and 2 confirm the analysis reported in Sections 4 and 5, regarding the graphs sizes. 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 time-expanded, the reduced time-expanded, and DTM graphs, respectively.

The results of our experiments on the query time and the update time for the graph-based models are summarized in Table 5 and 6 respectively. The combination of the query algorithms with ALT require a preprocessing phase, whose requirements are reported in Table 4. In what follows we analyze and comment the results of Tables 5 and 6, first with respect to the query time, and then with respect to the update time.

Table 3
Timetable attributes: average transfer time (mins), average degree of adjacent stops/stations and percentage of transportation means (% of total).

$\mathcal{T}$	efz	ger	eur	bts	ks	bvb	Victoria	Madrid	Athens	Rome
transfer time	6.5	6.5	6.5	6.5	6.5	6.5	6.5	3	3	3
outdeg	2.2	2.7	2.7	2.5	2.7	2.3	1.1	1.4	1.2	1.3
bus				100%						95.6%
light rail	100%									4.4%

$\mathcal{T}$	Berlin	London	Switzerland	Sweden
transfer time	0.7	0.8	0.3	0.2
outdeg	2.7	1.2	2.4	2.3
bus	76%	98%	80%	92.7%
light rail	11%	2%	12%	1.0%
commuter rail	4%		7%	0.4%
subway			1%	2.2%
tram	9%			3.7%

Table 4
ALT-based preprocessing time and space requirements for all timetables.

$\mathcal{T}$	bts	ks	efz	bvb	ger	eur
Space (GB)	0.002	0.014	0.019	0.032	0.161	0.177
Time (mins)	0.002	0.011	0.019	0.026	0.162	0.191

$\mathcal{T}$	Victoria	Madrid	Athens	Rome	Berlin	London	Switzerland	Sweden
Space (GB)	0.019	0.082	0.183	0.253	0.629	1.4	2.6	9.4
Time (mins)	0.022	0.063	0.153	0.249	0.729	1.785	4.004	15.648

Table 5
Comparison between reduced time-expanded graphs and dynamic timetable graphs with respect to average query time.

Type	$\mathcal{T}$	Query (ms)			
		`TE-QH-ALT`	`TE-QH`	`DTM-QH-ALT`	`DTM-QH`
HaCon	bts	0.17	0.28	0.14	0.15
	efz	0.26	0.97	0.43	0.53
	ks	0.43	0.82	0.42	0.47
	bvb	1.02	1.77	1.51	1.79
	ger	1.21	4.48	1.53	4.08
	eur	1.61	5.13	1.58	4.32
GTFS	Victoria	1.16	1.43	2.12	2.39
	Madrid	3.05	8.14	5.77	9.63
	Athens	2.12	4.60	3.62	7.61
	Rome	2.94	7.37	4.54	9.75
	Berlin	6.88	18.34	12.17	27.63
	London	5.14	15.13	10.25	31.12
	Switzerland	24.13	37.22	48.19	86.83
	Sweden	29.98	59.10	55.27	99.58

7.2. *Timetable queries*

In order to test the performance of the three models, we evaluated the time required for answering a bunch of EAP queries on each timetable. 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 the query algorithms on each type of graph.

The query algorithms tested in the experiments are: `TE-Q`, `TE-QH`, `TE-QH-ALT`, `DTM-Q`, `DTM-QH` and `DTM-QH-ALT`, respectively.

Preliminary experiments revealed that: (i) the performance of `TE-QH` is always better than that of `TE-Q` (in accordance with [31]); (ii) `DTM-QH` is always faster than `DTM-Q`. For these reasons, in Tables 5 and 6 we report only the results of `TE-QH`, `TE-QH-ALT`, `DTM-QH` and `DTM-QH-ALT`, respectively. We performed analogous experiments for MNTP queries, but we omit the results as they lead to similar analysis.

First of all, our experiments show that: (i) combining ALT with the restricted node exploration leads to a significant speed-up in query time with respect to the plain query algorithm, for both models; (ii) DTM query time is competitive to that of the reduced time-expanded model. Regarding this latter result, notice that part of the overhead, with respect to the

Table 6
Comparison between reduced time-expanded graphs and dynamic timetable graphs with respect to average update time. We also report the average number of affected arcs, that is, the arcs updated (weight change and/or rewiring) by `TE-UH` as well as the affected elementary connections (a proxy for the number of node time references) that need to be updated by `DTM-U`.

Type	$\mathcal{T}$	Update (µs)			
		TE-UH		DTM-U	
		time	aff. arcs	time	aff. arcs
HaCon	bts	34.47	37.7	2.19	8.8
	efz	25.31	30.9	2.25	7.2
	ks	39.49	53.8	3.04	11.1
	bvb	95.98	63.5	15.17	13.5
	ger	40.32	32.2	7.17	7.6
	eur	43.10	33.5	8.07	9.1
GTFS	Victoria	96.36	64.7	14.31	10.2
	Madrid	124.50	64.5	29.45	11.1
	Athens	424.74	181.0	46.98	59.4
	Rome	381.47	107.3	49.73	81.4
	Berlin	194.54	61.6	80.23	9.9
	London	477.23	130.4	271.46	29.0
	Switzerland	678.92	55.5	356.15	40.3
	Sweden	604.77	85.5	293.62	20.0

query time, of DTM is probably due to the fact that there are no separated arrival to departure arcs. In fact, for instance, during a query (S, T, t_S), in order to find the next valid departure node (i.e. having departure time greater than t_S), there is a need for searching among many departure nodes (possibly all), as they are stored in increasing order of arrival time instead of departure time. This makes the algorithm more time-consuming with respect to the reduced time-expanded model, where arrival to departure arcs allow to perform directly a binary search within the set of departure nodes (ordered by departure time). Therefore, the arrival node contraction in DTM results in this disadvantage.

Another outcome of our experiments is that all the query algorithms of DTM, i.e. `DTM-Q`, `DTM-QH` and `DTM-QH-ALT`, are much faster with respect to previously proposed versions, as they are the result of a further fine tuning. We refer the reader to [10] for the query times of previous versions of these algorithms. By comparing such results with those of Table 5, it is easy to see that: (i) the new version of `DTM-QH` is up to 50% faster than the previous one; (ii) the new version of `DTM-QH-ALT` is up to 30–35% faster than the previous one. Moreover, since 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 [31], our experiments show that the query time of DTM is comparable to that of the time-dependent model.

The query time depends on some critical transportation attributes, like the number of the adjacent stops and the number of the different transportation means. Based on Table 3, Sweden, Switzerland and Berlin network instances have greater density in those terms, resulting in a more expensive shortest path routing computation unlike in the other instances.

Our data also show that the query times of graph-based models are competitive with respect to those of array-based approaches. For instance, the query time of CSA is 2 ms on a snapshot of the network of London with around 5 millions connections [3], while the query time of `TE-QH-ALT` and `DTM-QH-ALT` is 5.14 ms and 10.25 ms, respectively, on a snapshot of the network of London with 14 millions connections (i.e., almost three times larger).

7.3. Timetable updates

In order to evaluate the performance of the update algorithms, 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 `DTM-U` for updating the dynamic timetable graph (presented in Section 5.3), as well as `TE-UH` for updating the reduced time-expanded graph (presented in Section 4.3). We measured the average computational time and the number of connections affected by the change, that is the number of connections associated to the same train or bus which has experienced the delay. We omit the results regarding the update routine of the realistic time-expanded graphs, i.e. `TE-U`, as preliminary experiments we conducted showed that the reduced time-expanded update routine is always at least 50% faster than the original one.

The experimental results are shown in Table 6. In this case `DTM-U` outperforms `TE-UH` with respect to the update time. The results confirm that the upper bound given in Section 5 for the computational time of the update algorithm is really far from being realistic, thus making DTM suitable to be used in practice. In fact, even in the biggest networks (London, Sweden and Switzerland), the updating algorithm requires less than 360 µs. Moreover, only (at most) two arc weights and few node time references need to be changed in the original graph to keep the EAP queries correct (in the tested timetables we measured at most 81.4 affected elementary connections on average). 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 $|\mathcal{B}|$.

Table 7
Query times required by graph-based models and by RAPTOR on the public transportation system of London (around 14 M elementary connections).

	TE-QH	TE-QH-ALT	DTM-QH	DTM-QH-ALT	RAPTOR
Query (ms)	15.13	5.14	31.12	10.25	7.79

7.4. Comparison with RAPTOR

For the sake of completeness, we directly compared the query time of our graph-based models with one of the fastest array-based algorithms that does not rely on a preprocessing phase, i.e. Round-Based Public Transit Routing (in short, RAPTOR), proposed in [17].

RAPTOR is explicitly developed for public transit networks and its basic version optimizes arrival time and the number of transfers taken. Instead of using a graph, it organizes the input as a few simple arrays of trips and routes. Note that, a *trip* is a sequence of stations that are connected by elementary connections served by the same vehicle, while a *route* is a set of trips that share the same set of stations.

Essentially, RAPTOR is a dynamic program: it works in rounds, with round i computing earliest arrival times for journeys that consist of exactly i transfers. Each round takes as input the stops whose arrival time improved in the previous round (for the first round this is only the source stop). It then scans the routes served by these stops. To scan route r, RAPTOR traverses its stops in order of travel, keeping track of the earliest possible trip (of r) that can be taken. This trip may improve the tentative arrival times at subsequent stops of route r. Notice also that RAPTOR scans each route at most once per round, which is very efficient in practice. Moreover, RAPTOR can be parallelized by distributing non-conflicting routes to different CPU cores. For a more detailed description of the algorithm and its performance, we refer the reader to [19].

The comparison between graph-based models and RAPTOR is reported in Table 7. In particular, we report, for each model the average query time achieved with a snapshot of the local public transportation system of London made of about 14 M elementary connections. Unfortunately, we are not able to directly evaluate RAPTOR in terms of update time, since, concerning its performance in dynamic scenarios, there is no evidence in the literature, and it is not clear how to apply/modify the RAPTOR code to handle dynamic updates to the timetable. In particular, in RAPTOR, in order to exploit data locality, both routes and trips are sorted within the arrays, with respect to time [19]. Therefore, any update occurring on the original timetable, in order to keep queries correct, would at least require: (i) to update the time associated to trips and/or routes; (ii) to check whether the corresponding entries, after the update, violate the ordering within the arrays; (iii) in that case, to move them in order to restore the ordering.

Table 7 shows that graph-based models are comparable to RAPTOR in terms of query time, and that the reduced time-expanded query algorithm, when equipped with ALT, is faster then RAPTOR.

Notice that for all graph based models, we considered their non-acyclic version, i.e., we do not unroll the timetable for several consecutive periods in order to handle overnight connections, as indeed is the case with array-based models. For this reason, we have to use a priority queue based algorithm for computing the shortest paths, such as the modifications of Dijkstra's we described in Sections 3.2, 4.2 and 5.2. This choice allows to us to save space at the price of losing something in terms of query time. Note that, a different trade off would result in better query time at the price of worse space occupancy.

8. Conclusions

In this work, we have studied the journey planning problem. We focused on the behavior of a prominent class of models, namely graph-based models, in the realistic case of dynamic scenarios, i.e., when the input timetable can (unpredictably) change subject to delays.

We made the following contributions: First, we have considered the graph-based *reduced time-expanded model* and have given a simplified and optimized routine for handling delays, and an engineered query algorithm. Second, we have proposed a new graph-based model, namely the *dynamic timetable* model, natively tailored to efficiently incorporate dynamic updates, along with a query algorithm and a routine for handling delays. Third, we have shown how to adapt the unidirectional ALT algorithm to such graph-based models, in order to improve the query algorithm performance. Finally, we have conducted an experimental study to assess the effectiveness of all proposed models and algorithms and to compare them with the array-based state of the art solution for the dynamic case, i.e., RAPTOR. We evaluated both new and existing approaches by implementing and testing them on real-world timetables subject to synthetic delays. Our experimental results have shown that: (i) the dynamic timetable model is the best model in terms of computational time required for handling delays; (ii) graph-based models and RAPTOR are comparable in terms of query time; (iii) the dynamic timetable model compares favorably with the reduced time-expanded model from the space occupancy point of view; (iv) combining the graph-based models with some speed-up techniques designed for road networks, such as ALT, is a very promising approach.

We plan to extend the current work 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 DTM. To this regard, we believe that footpaths can be easily incorporated to our graph based models in an efficient way. In fact, in the time-expanded models, if there is a footpath from stop p_i to stop p_j, then for each arrival event at stop p_i one adds an arc to the earliest reachable transfer vertex at p_j. On the contrary, in the dynamic timetable model, it is enough to slightly modify the query algorithm as follows: when a switch node of a certain stop p_i is extracted, if there is a footpath to a neighboring stop p_j, then the arrival time to p_j via a departure node simply has to be compared with the arrival time to p_j via the footpath, which is given by the sum of the arrival time to p_i plus the weight of the footpath, mod 1440.

There are several other directions for future work. One is that of combining other known speed-up techniques to the tested graph-based models. To this aim, the most promising ones are those based on Arc-flags [27] as they can be combined with ALT [16] and support dynamic updates [12]. Furthermore, the efficient implementation of ALT given in [22] could further improve the query time. Given these combinations, it would be also interesting to experimentally compare the models studied in this paper with other array-based models and time-dependent approaches. Another future direction could be that of tackling multi-criteria problems.

Acknowledgement

We are grateful to Thomas Pajor for providing us with the implementation of the RAPTOR algorithm.

References

[1] H. Bast, E. Carlsson, A. Eigenwillig, R. Geisberger, C. Harrelson, V. Raychev, F. Viger, Fast routing in very large public transportation networks using transfer patterns, in: 18th Annual European Symposium on Algorithms, ESA2010, in: Lecture Notes in Computer Science, vol. 6346, Springer, 2010, pp. 290–301.

[2] H. Bast, J. Sternisko, S. Storandt, Delay-robustness of transfer patterns in public transportation route planning, in: 13th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems, ATMOS2013, Schloss Dagstuhl, 2013, pp. 42–54.

[3] H. Bast, D. Delling, A.V. Goldberg, M. Müller-Hannemann, T. Pajor, P. Sanders, D. Wagner, R.F. Werneck, Route planning in transportation networks, in: Algorithm Engineering—Selected Results and Surveys, in: Lecture Notes in Computer Science, vol. 9220, 2016, pp. 19–80.

[4] R. Bauer, D. Delling, P. Sanders, D. Schieferdecker, D. Schultes, D. Wagner, Combining hierarchical and goal-directed speed-up techniques for Dijkstra's algorithm, ACM J. Exp. Algorithmics 15 (2010), Article 2.3.

[5] R. Bauer, D. Delling, D. Wagner, Experimental study of speed up techniques for timetable information systems, Networks 57 (1) (2011) 38–52.

[6] F. Bruera, S. Cicerone, G. D'Angelo, G.D. Stefano, D. Frigioni, Dynamic multi-level overlay graphs for shortest paths, Math. Comput. Sci. 1 (4) (2008) 709–736.

[7] S. Cicerone, G. D'Angelo, G.D. Stefano, D. Frigioni, A. Navarra, Delay management problem: complexity results and robust algorithms, in: Proceedings 2nd International Conference on Combinatorial Optimization and Applications, COCOA2008, in: Lecture Notes in Computer Science, vol. 5165, Springer, 2008, pp. 458–468.

[8] S. Cicerone, G. D'Angelo, G. Di Stefano, D. Frigioni, A. Navarra, Recoverable robust timetabling for single delay: complexity and polynomial algorithms for special cases, J. Comb. Optim. 18 (3) (2009) 229–257.

[9] S. Cicerone, G. D'Angelo, G. Di Stefano, D. Frigioni, A. Navarra, M. Schachtebeck, A. Schöbel, Recoverable robustness in shunting and timetabling, in: Robust and Online Large-Scale Optimization, in: Lecture Notes in Computer Science, vol. 5868, Springer, 2009, pp. 28–60.

[10] A. Cionini, G. D'Angelo, M. D'Emidio, D. Frigioni, K. Giannakopoulou, A. Paraskevopoulos, C.D. Zaroliagis, Engineering graph-based models for dynamic timetable information systems, in: 14th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems, ATMOS2014, in: OASICS, vol. 42, Schloss Dagstuhl, 2014, pp. 46–61.

[11] G. D'Angelo, M. D'Emidio, D. Frigioni, C. Vitale, Fully dynamic maintenance of arc-flags in road networks, in: Proceedings 11th International Symposium on Experimental Algorithms, SEA 2012, in: Lecture Notes in Computer Science, vol. 7276, Springer, 2012, pp. 135–147.

[12] G. D'Angelo, M. D'Emidio, D. Frigioni, Fully dynamic update of arc-flags, Networks 63 (3) (2014) 243–259.

[13] D. Delling, D. Wagner, Landmark-based routing in dynamic graphs, in: 6th Workshop on Experimental Algorithms, in: Lecture Notes in Computer Science, Springer, 2007, pp. 52–65.

[14] D. Delling, R.F. Werneck, Faster customization of road networks, in: 12th Symposium on Experimental Algorithms, SEA2013, in: Lecture Notes in Computer Science, vol. 7933, Springer, 2013, pp. 30–42.

[15] D. Delling, K. Giannakopoulou, D. Wagner, C. Zaroliagis, Timetable Information Updating in Case of Delays: Modeling Issues, Tech. Rep. ARRIVAL-TR-0133, ARRIVAL Project, 2008.

[16] D. Delling, T. Pajor, D. Wagner, Engineering time-expanded graphs for faster timetable information, in: Robust and Online Large-Scale Optimization, in: Lecture Notes in Computer Science, vol. 5868, Springer, 2009, pp. 182–206.

[17] D. Delling, T. Pajor, R.F. Werneck, Round-based public transit routing, in: 14th Workshop on Algorithm Engineering and Experiments, ALENEX2012, SIAM, 2012, pp. 130–140.

[18] D. Delling, J. Dibbelt, T. Pajor, R. Werneck, Public transit labeling, in: 14th International Symposium on Experimental Algorithms, SEA2015, in: Lecture Notes in Computer Science, vol. 9125, Springer, 2015, pp. 273–285.

[19] D. Delling, T. Pajor, R.F. Werneck, Round-based public transit routing, Transp. Sci. 49 (3) (2015) 591–604.

[20] J. Dibbelt, T. Pajor, B. Strasser, D. Wagner, Intriguingly simple and fast transit routing, in: 12th Symposium on Experimental Algorithms, SEA2013, in: Lecture Notes in Computer Science, vol. 7933, Springer, 2013, pp. 43–54.

[21] J. Dibbelt, B. Strasser, D. Wagner, Customizable contraction hierarchies, in: 13th Symposium on Experimental Algorithms, SEA2014, in: Lecture Notes in Computer Science, vol. 8504, Springer, 2014, pp. 271–282.

[22] A. Efentakis, D. Pfoser, Optimizing landmark-based routing and preprocessing, in: 6th ACM SIGSPATIAL Int. Work. on Computational Transp. Science, ACM, 2013.

[23] D. Firmani, G.F. Italiano, L. Laura, F. Santaroni, Is timetabling routing always reliable for public transport?, in: 13th Work. on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems, ATMOS2013, Schloss Dagstuhl, 2013, pp. 15–26.

[24] M. Fischetti, D. Salvagnin, A. Zanette, Fast approaches to improve the robustness of a railway timetable, Transp. Sci. 43 (3) (2009) 321–335.

[25] A. Goldberg, C. Harrelson, Computing the shortest path: A^* search meets graph theory, in: ACM-SIAM Symposium on Discrete Algorithms, SODA2005, SIAM, 2005, pp. 156–165.

[26] HaCon—Ingenieurgesellschaft mbH, http://www.hacon.de, 2008.

[27] U. Lauther, in: An Extremely Fast Exact Algorithm for Finding Shortest Paths in Static Networks with Geographical Background, vol. 22, 2004, pp. 219–230.

[28] C. Liebchen, M. Schachtebeck, A. Schöbel, S. Stiller, A. Prigge, Computing delay resistant railway timetables, Comput. Oper. Res. 37 (5) (2010) 857–868.

[29] G. Mali, P. Michail, A. Paraskevopoulos, C. Zaroliagis, A new dynamic graph structure for large-scale transportation networks, in: 8th International Conference on Algorithms and Complexity, CIAC2013, in: Lecture Notes in Computer Science, vol. 7878, Springer, 2013, pp. 312–323.
[30] M. Müller-Hannemann, M. Schnee, Efficient timetable information in the presence of delays, in: Robust and Online Large-Scale Optimization, in: Lecture Notes in Computer Science, vol. 5868, Springer, 2009, pp. 249–272.
[31] E. Pyrga, F. Schulz, D. Wagner, C. Zaroliagis, Efficient models for timetable information in public transportation systems, ACM J. Exp. Algorithmics 12 (2.4) (2008) 1–39.
[32] M. Schachtebeck, A. Schöbel, To wait or not to wait—and who goes first? Delay management with priority decisions, Transp. Sci. 44 (3) (2010) 307–321.
[33] D. Schultes, P. Sanders, Dynamic highway-node routing, in: 6th Workshop on Experimental Algorithms, WEA2007, in: Lecture Notes in Computer Science, Springer, 2007, pp. 66–79.
[34] Transit feeds, https://transitfeeds.com.
[35] Transport for London, https://tfl.gov.uk.
[36] D. Wagner, T. Willhalm, C.D. Zaroliagis, Geometric containers for efficient shortest-path computation, ACM J. Exp. Algorithmics 10 (1.3) (2005) 1–30.
[37] S. Witt, Trip-based public transit routing, in: 23rd European Symposium on Algorithms, ESA2015, in: Lecture Notes in Computer Science, vol. 9294, Springer, 2015, pp. 1025–1036.