Project Number FP6-021235-2

ARRIVAL
Algorithms for Robust and online Railway optimization:
Improving the Validity and reliability of Large scale systems

STREP
Member of the FET Open Scheme

ARRIVAL – TR – 0133

Timetable Information Updating in Case of Delays: Modeling Issues

Daniel Delling, Kalliopi Giannakopoulou, Dorothea Wagner, and Christos Zaroliagis

January 2008
Abstract. The timetable information problem can be solved by computing shortest paths in special graphs built from timetable data. In general, two models exist: the time-dependent and time-expanded network. In a recent work, both models are compared with respect to advantages and disadvantages on a theoretical and a practical framework. In addition, an extensive experimental evaluation reveals further differences with respect to query performance. However, delays—which occur very frequently in railway systems—are not covered. In this work, we show how the time-dependent and the time-expanded models should be updated in order to capture delays. It turns out that delays can be incorporated in the time-dependent model without changing the topology of the network. This is not true for the time-expanded model, whose updating involves a (sometimes large) sequence of edge insertions, deletions, and cost modifications.

1 Introduction

Finding timetable information is central to any public transportation system. The problem consists in modeling timetable information so that subsequent queries asking for optimal itineraries can be answered efficiently. In this work, we are concerned with a specific, query-intensive scenario arising in public railway transport, where a central server is directly accessible to any customer either through terminals in train stations or through a web interface, and has to answer a huge number of on-line queries. The ultimate goal is to answer such a query as fast as possible.

The aforementioned problem, called the static timetable information problem, is well studied and in a recent work [9] an extensive evaluation of various solution strategies for solving it has been presented. This evaluation includes a comparison of two approaches that model timetables as graphs; namely, the time-dependent and the time-expanded model. Both graphs are built from timetable data, and are directed and weighted. Queries are then answered by computing shortest paths in these graphs, typically using DIJKSTRA’s algorithm or variations of it, also known as speed-up techniques (see e.g., [10, 11] for an overview over speed-up techniques). In [9], several (basic) speed-up techniques for DIJKSTRA’s algorithm have been experimentally evaluated on both models.

Despite how well the various components of a railway system have been planned ahead, delays occur inevitably for several reasons. As a result, the (actual) timetable may be altered and the queries issued by customers should reflect the new situation. This brings us to the dynamic version of the timetable information problem, where the main issue is how to efficiently update the timetable graphs so that subsequent queries are answered correctly. All previous studies, including [9], do not deal with the dynamic timetable information problem.
At a glance, one might expect that delays can be incorporated easily by simply updating the travel time of the delayed train. While this turns out to be true for the time-dependent model, things are more complicated for the time-expanded model. In the latter, nodes represent time events (departures, transfers, and arrivals) within a day and edge weight updating is not sufficient, as it is demonstrated by the example of Figure 1. The leftmost graph of Figure 1 models the case without any delay (the precise details of the modeling will be explained in Section 2), where train $\alpha$ arrives at a station and its passengers can transfer to trains $\beta$ and $\gamma$, or continue with the same train ($\alpha'$). Now let train $\alpha$ be delayed by 10 minutes. By simply increasing the edge weight by 10 minutes (center graph), timetable information queries would be wrong: train $\beta$ is still reachable according to the graph. However, this is not true (unless $\beta$ waits). Thus, we also have to update and “rewire” station edges as well (rightmost graph). With these updates $\beta$ is not reachable any more.

This edge updating process propagates to all subsequent stations through which the delayed train passes. These updates may even yield topology changes within the graph.

![Fig. 1. Delay of a train and its modeling.](image)

In this work, we make the first step towards the solution of the dynamic timetable information problem by discussing modeling issues. In particular, we show how the time-dependent and time-expanded graph models should be updated in order to capture delays. It turns out that delays can be incorporated in the time-dependent model without changing the topology of the graph. This is not true for the time-expanded model, whose updating involves a (sometimes large) sequence of edge insertions, deletions, and cost modifications, due to the propagation of the delays to the whole railway network. This may have a cascading effect: it is not only the delayed trains that have to be accommodated, but also other trains that (according to the rail operator policy) should wait for the delayed ones, and hence introducing further delays to the system.

The rest of the paper is organized as follows. In Section 2, we shortly recapture the models for the static timetable information problem. In Section 3, we present our approaches for updating the time-dependent and the time-expanded timetable graph models. Our work is concluded by a summary and future work in Section 4.

## 2 Modeling Timetable Information

In this section, we briefly summarize existing approaches to model static timetable information as graphs (cf. [9] for details). We hereby concentrate on realistic models, i.e., models that incorporate transfer times.
2.1 The Time-Dependent Model

The time-dependent model is an efficient and compact approach to model timetable information. A node is introduced for each station and an edge is inserted iff a direct connection between two stations exist. Several weights are assigned to each edge. Each weight represents the travel time of a train running from one station to another. More precisely, we use link-traversal weights [3]. At each edge we store a function that returns the arrival time at the target of the edge according to the departure time from the source. See Fig. 2 for an example. The advantage of this model is its small size and the obtained travel time is feasible. However, transfer times within stations are not incorporated. Due to this, this model is called the simple time-dependent model.

![Weight of a time-dependent edge.](image)

As already stated, we do want to incorporate transfer times, which are part of the realistic time-dependent model. Again, edge-weights are time-dependent, but a station is modeled by more than one node. More precisely, each possible route of a train is modeled separately. Then, transfer edges are added within the station. In [9], two scenarios are presented. In a scenario with constant (e.g., maximum) transfer times per station, non-time-dependent edges are used for modeling transfers. Therefore, an extra station node is added. An edge from each route node to the station node is inserted having a weight of 0. The opposite edge has a weight of the transfer time at the station. Figure 3 gives an example.

Sometimes, transfers can highly vary. Imagine you just arrived at a big station. A train departing at the same platform can be boarded much faster than a train departing at the other side of the station. Moreover, as we use train routes over a day, the departure platform may change during the day. In order to incorporate such details, the time-dependent model with variable transfer times connects route nodes within a station with time-dependent edges representing the transfer times between both routes during the day. Figure 4 gives an example.

2.2 The Time-Expanded Model

One potential disadvantage of the time-dependent model is its applicability for speed-up techniques for Dijkstra’s algorithm. The fastest techniques [4, 1] rely on non-time-dependent
Fig. 3. Time-dependent model with constant transfer times. 4 train routes exist at station A while 3 exist at station B. \( T_A \) is the transfer time at station A, \( T_B \) the corresponding time at station B. Edges between stations are time-dependent.

Fig. 4. Time-dependent model with variable transfer times. Unlike in Fig 3, edges within a station are also time-dependent.

edges. Thus, the time-expanded model “rolls out” time-dependency by modeling each station by several events yielding non-time-dependent edge weights. Each event represents the departure or the arrival of a station. In order to incorporate transfer times, transfer nodes are introduced as well. An example is given in Figure 5. As a result, 4 types of edges exist in the realistic model: connection, transfer–departure, arrival–departure, and arrival–transfer edges.

- The connection edges are inserted between departure and arrival events and correspond to a real connection between the stations. The weight of these edges is the real travel time of this connection.
- For each departure event, a transfer event is inserted to the graph. The according transfer–departure edges have a weight of 0.
- As long as a train does not end at the specific station, an arrival–departure edge is inserted between the arrival of this train in this station and its departure. Due to the fact that passen-
gers may stay in the train, this departure event is the only one that can be reached without entering the station.

- In order to change trains at a station the train has to be left. Thus, from each arrival node an arrival-transfer edge to the next reachable transfer node (i.e., to a node corresponding to the first departure that a passenger can reach) is inserted.

Fig. 5. Time-expanded model. At each station three types of nodes exist. The train represented by the connections $\alpha$ and $\gamma$ stays for 2 minutes in station $B$. The transfer time $T_B$ at station $B$ is set to 3 minutes.

3 Modeling Delays

In the last section, we explained how to model timetables as graphs in order to solve the timetable information problem by running shortest path queries in the resulted graphs. However, the presented models do not incorporate delays which occur quite frequently in railway systems. Here, we show how the graphs have to updated when a train is delayed. We concentrate on the realistic models.

3.1 Delays in the Time-Dependent Model

Incorporating delays in the time-dependent model is easy. For a delayed train, we simply have to increase the arrival time at each station the train is going to stop. As we use the train-route graph, we simply follow the route to the end. For an example of the impact of a delay on time-dependent edges see Fig. 6. However, the big advantage in the this model is that we do not have to update transfer edges, thus we do not have to change the topology of the network.

3.2 Delays in the Time-Expanded Model

As already mentioned in the Introduction, and unlike the time-dependent model, it is not sufficient to simply increase the travel time of the delayed train in the time-expanded model. Instead, we have to update edges within stations as well. As a consequence, our update routine has three
steps. First, we update the connection edge, then we update the weights of arrival–transfer and transfer–departure edges at all subsequent stations through which the delayed train passes. Finally, we have to check for every updated arrival–transfer edge whether the update still yields valid transfer times, i.e., the edge weight is still bigger than the transfer time for this station. In the following, we explain each step separately.

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

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

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

Figure 7 gives an example for the result of this update routine in case of a delay $\Delta = 8$ minutes.

**Boarding Delayed Trains.** The above described updates miss one important fact. Due to a train being delayed, it may happen that new connections are possible. In Figure 7, passengers on train $\beta$ could easily catch connection $\gamma$. However, our update does not allow this connection as there is no path between the arrival of $\beta$ and the departure of $\gamma$. While this is often wanted by the station operator, it would be nice to allow these transfers.

In order to allow these transfers, we simply add some additional edges to the graph. For each possible new connection, we add new arrival–departure edges to the departure event of the delayed train. We therefore only have to take into account those connections that at latest depart $t = \Delta - T$ minutes after the scheduled arrival of the delayed train, where $\Delta$ is the delay of the train and $T$ is the transfer time within the station. An advantage of this approach is that we can manually adjust at which stations (and for which trains) we allow such connections. Figure 8 gives an example.
Fig. 7. Modeling delays in the time-expanded model. The graph shows the same example from Fig. 5 but train $\alpha$ is delayed by 8 minutes. The equivalent edge is increased by 8 minutes and the according arrival–transfer and transfer–departure edges are altered at all following stations. For station B, we have to remove the original arrival–transfer and add a new one, while for station C, it is sufficient to decrease the edge weight to 23.

Fig. 8. Allow new connections by delayed trains in the time-expanded model. As train $\alpha$ arrives later than $\beta$ due to a delay, a new departure–arrival edge is inserted at station B.

Waiting of Trains. Besides the fact that trains may be delayed by external incidents, e.g., suicide or train breakdowns, trains may also be delayed by waiting for other trains. Such so-called secondary delays can also be modeled in the time-expanded model. In principle, the waited train has a longer lease time than originally at a specific station and thus, it arrives later at the station he normally would stop. More precisely, we first increase the according arrival–departure edge of the waiting train. Next, we update the station edges within the station the train is waiting. As this train is now delayed by the additional waiting time, we have to update the following stations the train has to stop in a similar way like for the delay routine described above. In the following, we explain each step separately.

Step 1 (Increase Waiting Weight). We identify the train waiting for another train. The according arrival–departure edge is increased by the additional waiting time $\Delta$ of the train.

Step 2 (Update Station Edges at Waiting Station). Next, we increase the weight of the transfer–departure edge of the waiting train by $\Delta$. This models the delayed departure of the train. As
this delayed departure allows additional transfers, we add arrival–departure edges to the graph that model such transfers.

**Step 3 (Update Station Edges at Following Stations).** For all subsequent stations the waiting train stops we have to update both transfer–departure and arrival–transfer edges. For the former we simply increase its weight from 0 to the delay \( \Delta \), while for the latter we decrease each arrival–transfer edge by \( \Delta \).

**Step 4 (Validate Station Edges at Following Stations).** Our final step checks for every altered arrival–transfer edge, whether these edges are still valid, i.e., the edge weight is still bigger than the transfer time for this station. In case an edge is valid, we are done for this edge, but in case not, we have to rewire the edge. The target node has to be changed to the next reachable transfer node, i.e., the first node resulting in a valid arrival–transfer edge.

Note that step 3 and 4 equal steps 2 and 3 from our delay routine described above. See Figure 9 for an example.

Fig. 9. Modeling waiting in the time-expanded model. The graph shows the same example from Fig. 7 but train \( \beta \) is waiting 4 minutes, so that delayed passengers from \( \alpha \) can board \( \beta \) (recall that the minimum transfer time at station B is 3 minutes). The arrival–departure and transfer–departure edge is increased by 4 minutes and an additional edge is inserted. Note that we additionally have to update the station edges of all stations \( \beta \) is stopping.

### Conclusion and Outlook

In this work, we investigated the modeling issues of the dynamic timetable information problem. In particular, we showed how existing approaches for timetable information should be updated so that delays can be incorporated. It turns out that updates are somehow more complicated for time-expanded graphs as the topology of the network is altered when delays occur.

As next logical step, we plan to investigate how existing speed-up techniques can be adapted to these dynamic scenarios. The most promising candidate is the ALT algorithm [6], which is proven to work well in scenarios where small updates occur frequently [5]. Moreover, ALT harmonizes well with Reach [7, 8], which can be used in dynamic scenarios as well [2]. Thus, it may be promising to use Reach in combination with ALT in order to drastically accelerate dynamic timetable information queries.
References


A Alternative Propagation of Delays

Another important parameter is that some trains wait and some others do not wait in case of a delayed train. We define as waited the train that waits for a delayed train, and no waited otherwise. Waited trains may be the source of additional delays, causing the propagation of delays to the whole system. In this section, we describe how this delay propagation is modeled.

We assume that within a certain time-frame a delay occurs in only one train in a station, and that at least one other train waits in the same station after the delayed train (see Fig. 10) – otherwise, we come to the case of Section 3.2.

We describe the modeling of delay propagation in three phases. The first phase refers to the delayed train, the second one to the waited trains, and the third one to the other trains. In the following, we explain each phase separately.

**Delayed Train.** Let train $\beta$ be the delayed train and let it arrive at a station after a delay of $\Delta$ minutes; see Fig. 11.

1. Delete the arrival-transfer edge $(a_{\beta}, t_{\beta})$, the arrival-departure edge $(a_{\beta}, d_{\beta})$, and insert a new arrival-transfer $(a_{\beta}, y)$ for this train with the weight $\Delta$.
2. Insert a new edge $(y, t_{\beta})$ from the transfer node $y$ of the delayed train (representing the first possible departure for which the passengers of train $\beta$ can make a transfer) to the original transfer node $t_{\beta}$ with zero weight.

**Waited Trains.** Let train $\gamma$ be a waited train that waits for the delayed train $\beta$; see Fig. 12.

1. Delete the arrival-departure edge $(a_{\gamma}, d_{\gamma})$.
2. Delete the transfer-transfer edge $(t_1, t_2)$, which is prior to transfer node $t_2$ of the waited train and insert a bypass edge to the first no-wait transfer node $t_k$. The weight $z$ of this edge will be the sum of the weights of all edges until the transfer nodes between $t_1$ and $t_k$ i.e., $z = z_1 + z_2 + \ldots + z_{k-1}$. 
Fig. 10. Time-expanded network. If train $\beta$ has a delay, then the delay policy dictates that trains $\gamma$ and $\epsilon$ wait for $\beta$.

Fig. 11. The modeling of the Delayed Train. Train $\beta$ arrives with $\Delta$ minutes of delay.

3. Insert a new transfer-transfer edge $(t_k, t_2)$ from the proper transfer node $t_k$ to transfer node $t_2$ with zero weight.

Other Trains. The other trains are the trains which either have connections with the delayed train, or have connections with the waited train. Both of them are modeled similarly, as follows (see Fig. 13).
1. Delete the arrival-transfer edge \((a_\alpha, t_\beta)\).

2. Insert a new arrival-transfer \((a_\alpha, t_y)\) edge to the next transfer node of the delayed or the waited train with the weight \(w = w_1 + z_1\).
Example. Fig. 14, illustrates the modeling of the propagation of the delay of train $\beta$ in Fig. 10. The delayed train has $\Delta = 42$ minutes of delay, and we have two waited trains $\gamma$ and $\epsilon$. Train $\alpha$ is a train that has a connection with train $\beta$, and train $\delta$ has a connection with train $\epsilon$, which has to wait for the delayed train $\beta$.

**Fig. 14.** The modeling of delay propagation.