Deriving all Passenger Flows in a Railway Network from Ticket Sales Data

Peter Sels [1,2,3], Thijs Dewilde[1], Dirk Cattrysse[1], Pieter Vansteenwegen[4]

[1]Katholieke Universiteit Leuven,
Centre for Industrial Management/Traffic & Infrastructure,
Celestijnenlaan 300a, 3001 Heverlee, Belgium

[2]Logically Yours BVBA,
Plankenbergstraat 112 bus L7, 2100 Antwerp, Belgium
e-mail: sels.peter@gmail.com, corresponding author

[3]Infrabel, Department of Network Access,
Frankrijkstraat 91, 1070 Brussels, Belgium

[4]Ghent University, Department of Industrial Management,
Technologiepark 903, 9052 Zwijnaarde, Belgium

Abstract
In optimizing the national railway timetable for the Belgian infrastructure company Infrabel, we use the quality criterium of minimizing expected total passenger travel time. Optimizing the timing of all trains fairly over all passengers requires passenger flow information. Focus is usually on determining the *train service offer side*, being the train timing, without considering the *demand side* data, being the passenger flow information. In this paper, we focus on passenger flow derivation.

We start from train ticket sales data and use Dijkstra's modified shortest path algorithm on the Belgian passenger train graph to determine a route per passenger origin destination station pair. We sum route passenger flows over all these routes. We also show how, in optimization of a timetable, timing and flows should be treated as interdependent. This means that we can use a process of iteration over these two phases, which we then call *reflowing* and *retiming*.

The infrastructure company did not have this passenger flow information available for the current schedule, but they need it to optimize many decisions. We determined these passenger flows for the current timetable.

We also give some recommendations regarding the improvement of passenger flow related data recording procedures, which would significantly improve passenger service.

Keywords
Railway timetabling, Passenger service, Passenger flows, Shortest path routing, Dijkstra

1 Introduction

In the context of setting up a *good* national railway timetable for the fixedly treated lines of Belgian passenger trains, the question arises as to what *quality criterium or criteria* should be used. For a client driven organization, asking the passengers is the logical answer. In questionnaires, this - not surprisingly - consistently points to lower total travel times as number one on their wish list. Travel times can be seen as average times or worst case times, but the variation in times is also required to be low. Low variation is a property of a robust schedule. To objectively evaluate any of these criteria, total passenger time is to be summed over all passengers, each one having the same weight. For this to be possible, we need to know passenger numbers on any unit piece of their trips. Rather surprisingly, it turns out, that this data is not available. A chip card system would give this information but is not yet implemented in Belgium. Yearly, partial counts are performed, but this generates too little information to be usable. This means that up to now, no such objective quality criterium was used in optimizing a railway schedule in Belgium. We need these passenger numbers, so we set up a procedure to derive them from ticket sales and some other data. Later, we give some recommendations for data collection that could greatly improve results, and are easy to perform.

In Section 2, we introduce the train network graph, how the reflowing and retime phase work on this graph, and how iteration over them works and decreases the total expected passenger time. Section 3 explains the hourly asymmetry inherent to real passenger flows, the lack thereof in the available data and how data collection can be improved. Section 4 describes how zone to zone based ticket sales numbers can be converted to approximate station to station based data and again how data collection can even improve its results. Section 5 describes how, from these station to station passenger numbers, passenger routes and the resulting flows can be derived. It also shows how this procedure can be used on an existing as well as a still to be optimized train timetable. We give a practical algorithm, verify its output data where we can, and show what is new and useful about this data. Section 6 sketches how the problem scope should be extended when the effect of a change in Origin-Destination-matrix has to be included. Section 7 summarizes the main contributions of this paper while Section 8 mentions our plans for related further work.

2 The FAPESP Problem

In formalizing the problem and the graph to model it, we are inspired by the formulation of the timetable problem as a Periodic Event Schedule Problem (PESP) [7, 8, 10, 11, 12, 15, 16]. This is the problem of determining good or optimal values for periodic event times or equivalently the durations between these event times. Since PESP does not consider (passenger) flow variables, we need to extend it. We propose to call the extended problem FAPESP, for Flow Allocation and Periodic Event Scheduling Problem. Comparable research has been performed by Bouma et. al. [1], Hofker et al. [9] and Exel et. al. [6] and led to the tools PROLOP and TRANS, developed by QQQ Delft, for passenger flow derivation on the Dutch train network. The graph we set up is similar to the PESP graph, but apart from a duration variable for each edge, it also has a flow variable for each edge. We describe and construct all objects in detail here, which will lead us to the final FAPESP graph.

2.1 The FAPESP Graph

Train and Passenger Actions
To be able to let passengers choose their preferred routes present in the train network, we need to model this network. This network models the *train* service. All trains intermittently ride and dwell between and at stations. Also, we have *passengers* realizing transfers between trains. This results in three types of actions. The train actions *ride* and *dwell* and the passenger *transfer* action. Every action has one passenger flow. So these are the values to be derived.

For the further optimization, it is noted that either the flow numbers on these separate actions will be used, or otherwise, flows per route will be used, where we need to be able to follow passenger flows per route, through their whole sequence of ride, dwell and transfer edges. This could be useful for example, if we need to evaluate the probability that a person following a certain route will arrive on time [13]. So we keep data-structures for both separate action flows and origin to destination routes with their associated flows.

Vertices and Edges
The set of actions can be practically represented by a directed graph $G(V, E)$ with vertex set V and edge set E. Refer to Figure 1 for construction of this graph.

Vertices come in two kinds for trains and two kinds for passengers.

- a train arriving at a station, set $V_{arr} \subset V$
- a train departing from a station, set $V_{dep} \subset V$
- a set of in passengers going to depart from and now entering a station, set $V_i \subset V$
- a set of out passengers having arrived at and now leaving a station, set $V_o \subset V$

where, $V_{arr} \cup V_{dep} \cup V_i \cup V_o = V$ and all pairwise sets are distinct.

Edges model the activities for which we want to determine flows and durations. They are all directed ones and come in two kinds for trains and three kinds for passengers:

- a train riding between its departing time at the previous station and its arrival time at the next station, set E_r
- a train dwelling between its arriving time at the current station and its departing time from the current station, set E_d
- a train turning around at its end station to go the other direction, set E_{ta}. The set E_{ta} may be considered a subset of the set E_d but in fact other constraints apply. For example passengers will not be able to enter a train when it is performing a turn around action. Also, a minimal service time for a turn around action will usually be larger than a minimal dwell time for a standard dwell action.
- a set of incoming passengers entering our system graph at a certain departing outbound train, set E_i
- a set of passengers transferring between a feeder train arriving in a station and a different, outbound train departing from that same station, set E_{tr}. Note that the transferring passenger can start his transfer at the beginning of the dwell action of

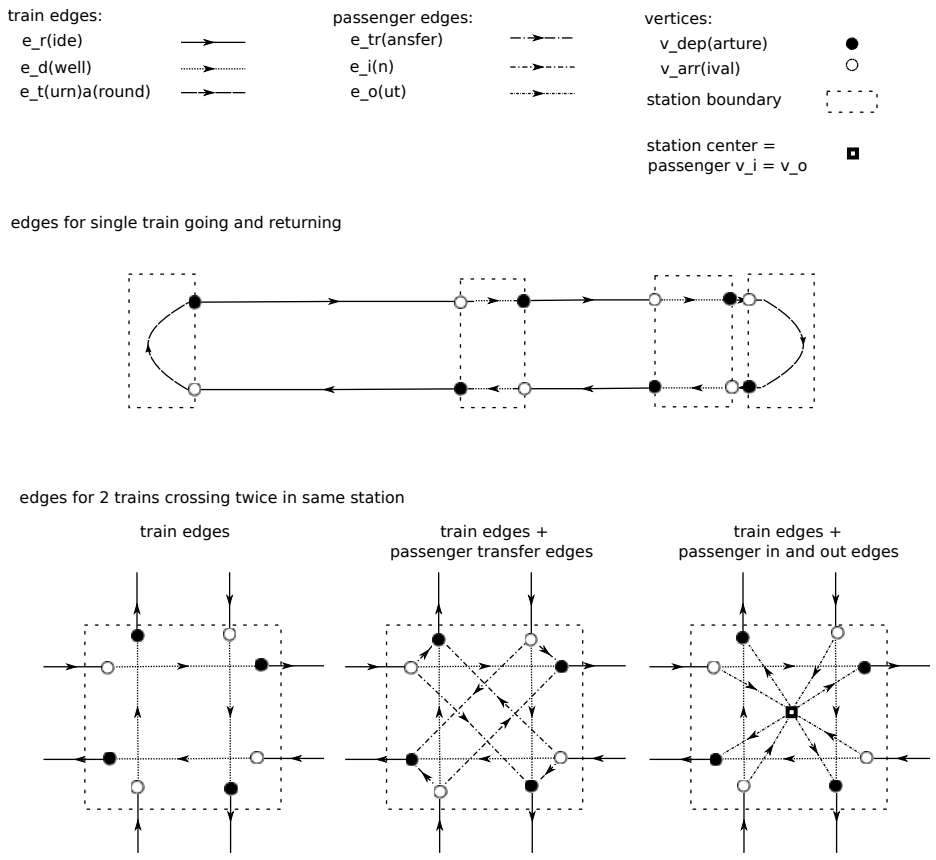

Figure 1: FAPESP EdgeTypes

the feeder train and should end transferring before the end of the dwell action of the departing train. Because an edge represents the maximum allowed time for its action, we model the transfer edge between these points.

- a set of outgoing passengers leaving our system graph at a certain departing outbound train, set E_o

where again, $E_r \cup E_d \cup E_{ta} \cup E_i \cup E_{tr} \cup E_o = E$ and all pairwise subsets sets are distinct. Note that the in and out edge types are necessary if we want to distinguish passengers flows entering (or leaving) different trains at the same station.

2.2 The Two-Phased Approach

Reflow Phase
The task of the reflow phase will be to assign a passenger flow f_e to every edge $e \in E$. Note that the reflow phase does not do any assignment to any vertex variables. The input data here is the graph structure, including the edge duration d_e of each edge e, together with the number of passengers per origin station to destination station pair, called *OD-pair*. A shortest route will be chosen for each OD-pair. This is done here with Dijkstra's shortest path algorithm. Per edge e, flows from all routes that pass through e, can then be accumulated to obtain the total flow f_e of e.

Retime Phase
Similarly, the task of the retime phase will be to assign a duration d_e for every edge $e \in E$, but this time, also satisfying all infrastructure constraints. Currently, we consider the duration for all in and out edges from E_i and E_o equal to 0. The input data here, is again the graph structure, this time annotated with the flows f_e obtained in the previous phase: the reflow phase. These f_e are treated as fixed now. The variables are d_e for which we determine an ideal value here. There is of course a minimum to each d_e per action, which we represent by m_e. The technique we use to do this is Mixed Integer Linear Programming (MILP).

On the vertex level, the retime phase results in event times t_v per vertex $v \in V$ that are separated by the durations of the edges between them. But in fact this is only a derived property. Also, if we add a constant time to each vertex time t_v, the result is still a valid solution of exactly the same quality. With quality, we mean goal function value, which is discussed next.

Same Goal Function for Both Phases
The goal of both phases: reflowing and retiming, will be to choose f_e and d_e values respectively so that a *single* goal function is minimized. In this function the assignments f_e and d_e are the complete set of variables.

We can write this as

$$G : E^{|E|} \to \mathbb{R} : e \mapsto g(f_e, d_e), \tag{1}$$

where the goal function G does a mapping from all edges e, using its f_e and d_e values of the latest reflow and retime phases respectively to a real goal function value.

So, for the reflow phase, we derive the set f_e, supposing that the d_e set is constant while for the retime phase, oppositely, we derive the set d_e while supposing the set f_e to be constant.

Each phase only has an effect on one of the variable sets. But what is important towards global optimization, is that they share the *same* goal. This is taken care of by using the same goal function for both. We can then even iterate over these two consecutive phases. Note that an identical goal function, in general, does not guarantee yet that iteration of both phases will converge to a fixed point solution. Oscillation between different good or not so good yet solutions could still be possible in steady state.

The retime phase corresponds to the aforementioned PESP problem. Because, for retiming, our goal function has fixed f_e values, it depends only on d_e. Integer linear programming techniques are often used to solve PESP. This can be done if (1) is linear in d_e. As a more specific example of a linear goal function (in this case to be minimized) we propose to use the total expected passenger time

$$g(f_e, d_e) = \sum_{e \in E} f_e d_e. \tag{2}$$

Other functions that use the assignments f_e and d_e can be used though, without breaking the workings of the iterative system.

Using a stochastic formulation of the total expected passenger time, we can even include a balance between a speedy service and robustness and still remain linear. Constraints are linear too. This will be the topic of a future paper on retiming. Concerning this approach to robustness, we refer to the papers of Vansteenwegen et al. [17, 18] and Dewilde et al. [4].

2.3 Self Sufficient Iterations

We already mentioned that the reflow and retime phases can be executed iteratively in a sequential loop. Figure 2 shows this idea graphically. The rectangular boxes represent operations, while the rounded boxes show data in- and outputs or decision points. zod_{z_o,z_d} stands for the zone origin destination matrix which specifies for every origin zone to destination zone how many passengers we have. The Mapper in Section 4 will do a mapping from this to a station to station OD-matrix, called sod_{s_o,s_d}. FA stands for Flow Allocation, or the reflow phase, which results in initial values for f_e in the first iteration or more appropriate values for f_e adapted to the changed d_e values, in all subsequent iterations. In all iterations, the retime phase PESP, will produce better values for the edge durations d_e, adapted to these changed f_e flow values. Cordone and Radaelli [2] as well as Schmidt and Schöbel [14] also came to an approach of iterations of similar two steps. We call this FAPESP iteration a self sufficient one, because the reflow phase, by being able to model peoples behavior in selecting the shortest route, does not depend on external data.

What remains to be done now is to describe in detail what we mean by the data and operations in the rounded and rectangular boxes. This is the focus of the remainder of this paper. We start with the meaning of zod_{z_o,z_d} in theory versus practice.

3 Morning versus Evening Traffic Separation, Hourly Asymmetry and Hourly Sum

3.1 Hourly Asymmetry

In fact the data for zo_{z_o,z_d} we had available is entirely and exactly symmetric in their OD-pairs.

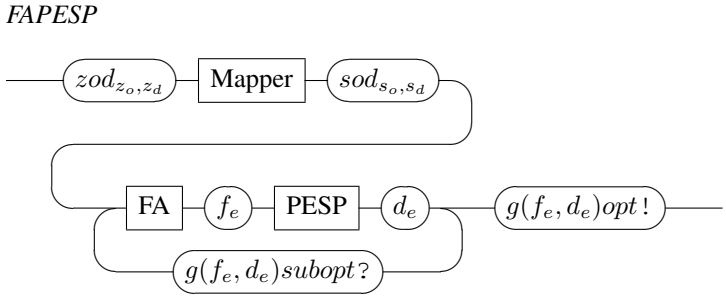

Figure 2: FAPESP Two Phase Flow Chart
offer and demand iteratively adapting to each other

This means that the number of passengers between zones A and B is always equal to the number of passengers between B and A. This was at first surprising, but is explained by the fact that it are sale figures of seasonal tickets, always valid in two directions between two zones.

However, this is unfortunate, since here information gets lost about the direction of travel. For instance, no procedure could possibly be able to separate out the passenger flows in the morning from the ones in the evening.

One could argue that this information is not needed, since timetables should be symmetric anyway. One of the reasons given is that *people who travel one way in the morning usually come back in the evening*. This is true, but not the correct reason. Symmetry is a property of a timetable, even within one period, usually one hour [11].

To know

$$\forall e \in E : d_e = d_{s(e)}, \tag{3}$$

where $s(e)$ is the symmetric edge of e. The symmetric edge for a train riding from station A to subsequent B is the edge for the opposite train going from station B to A. This implies that every train has another train in the opposite direction. But e can be any train or passenger edge type, ride dwell, transfer, in, out. What is important here is that both edges e and $s(e)$ make part of the same hourly instance of a the full days schedule.

However, within one hour, commuting flows are certainly not symmetric. Typically morning flows bring people from smaller places to bigger cities and oppositely the evening flow brings people from big cities to these smaller places again. So separately treated, morning passenger flows within each morning hour is not symmetric nor is the evening passenger flow. This means that, morning and evening separately treated again, we would need fewer trains per hour in one direction than in the other.

The main reason that still, per hour symmetric schedules are common, are rolling stock constraints.[8, 11] Indeed, if the frequency of trains from A to B is higher than backwards from B to A, material availability will increase in B and decrease in A. In fact this would be good to support the backwards flow in the evening, but this requires storage capacity and also leaves material sitting idle, which all have a cost too. These costs should in fact be weighted against the extra cost of requiring a symmetric schedule. For a clear description of benefits and costs of symmetry, we refer to Liebchen [11].

For now, missing data, working for short term and also considering that material handling cannot be immediately totally redesigned from scratch, we will keep using the sym-

metric simplification of the OD-matrix. But we will remember that using symmetric flow data misses a chance on a passenger service improvement of which the importance is yet unknown.

3.2 Separation Recommendation

As conclusion of the previous paragraph we can recommend an improvement to the data collection procedures for the train operator company.

At the source, let the train passengers specify their morning source and morning destination at ticket purchase time and keep accurate record of this. This would show the asymmetric nature of passenger flow data, which is important to keep, to be able to later, correctly balance passenger service benefits with material handling costs.

4 The Mapper: Mapping Zone Pair to Station Pair Numbers

We mentioned in 2.3 and showed in Figure 2, that we start from seasonal subscription ticket sales data to derive local flows on edges. Note that, this information represents the current customer OD-matrix and not the desired one, independent of the current train service offer. This corresponds in choice modelling with *preferred* versus *stated* preference. Also, we will see that this information is not yet in a directly usable format to start calculating routes for them in our graph, and how we can convert it first. This is the task of the mapper.

4.1 Zones instead of Stations

Tickets, in Belgium, are issued and sold, from one *origin zone* to a *destination zone*. A zone is a collection of geographically clustered stations. This means that from the ticket sales data, only the origin zone and destination zone are known, and not the specific respective stations. This is unfortunate. We will have no other choice than to somehow proportionally map zone to zone pairs to station to station pairs.

4.2 Zone to Station Matrix Conversion

From an Infrabel study, we had available the absolute numbers poM_s of departing passengers per origin station per Morning, based on actual physical countings. In these countings, passengers at the origin station were not asked what destination they were going to. So these countings only give a single number per origin which is the total for all the destinations corresponding with this origin.

To obtain the (departing) importance of a station in a zone, we derived all ratios of each stations' poM_s over the sum of all poM_s over the stations in their zone. Formally, if $zodM$ is the origin destination matrix expressed in zones for the Morning, and S_z the set of stations in the zone z, then the ratios roM_s standing for ratios of origin in the Morning are defined and calculated as

$$\forall (z_o, z_d) \in dom(zodM) : \forall s \in S_{z_o} : roM_s \equiv \frac{poM_s}{\sum_{s' \in S_{z_o}} poM_{s'}}, \tag{4}$$

where $dom(zod)$ represents the *domain* of the zone origin destination matrix zod, which

represents all the origin-destination zone pairs. (The range, $ran(zod)$ would be their respective passenger numbers.)

Similarly, for the arriving importance of a station in its zone, on the destination side we have

$$\forall(z_o, z_d) \in dom(zodM) : \forall s \in S_{z_d} : rdM_s \equiv \frac{pdM_s}{\sum_{s' \in S_{z_d}} pdM_{s'}}, \tag{5}$$

where pdM_s are the numbers of passengers arriving per destination station in the Morning, and rdM_s their correspondingly derived ratios. In fact the pdM_s were not counted and are not available, but are necessary for the explanation that follows. We will show later how we solve this problem.

With poM_s and pdM_s values, we could derive the station to station Morning OD-matrix $sodM$ from the zone to zone Morning OD-matrix $zodM$ as

$$\forall(z_o, z_d) \in zodM : \forall(s_o, s_d) \in (S_{z_o} \times S_{z_d}) : sodM_{s_o,s_d} \approx roM_{s_o} \times zod_{z_o,z_d} \times rdM_{s_d}. \tag{6}$$

As an example, suppose we have 100 passengers, going from one origin zone to a destination zone. We suppose now that the departing respectively arriving passengers are counted on origin and destination zone stations and give the poM_{s_o} and pdM_{s_d} numbers given in Table 1. This table also shows the sod_{s_o,s_d} numbers derived by (6).

Table 1: Derived Origin Stations to Destinations Station Matrix Example

		pdM_{s_d}	10	20	40	30
		rdM_{s_d}	0.1	0.2	0.4	0.3
poM_{s_o}	roM_{s_o}	$sodM_{s_o,s_d}$				
40	0.4		4	8	16	12
10	0.1		1	2	4	3
50	0.5		5	10	20	15

In fact, the situation is more complex. In practice, the roM_{s_o} ratios can be different across origin stations, resulting in different ratios per destination station for each origin station. This exactly represents the unrecoverable data loss resulting from being supplied a zone to zone matrix instead of a station to station matrix. Hence also the approximation sign in (6).

4.3 Lack of Destination Countings

As we mentioned in 4.2, only departing passengers were counted, and not the arriving ones, the pdM_{s_d} values. So we had to approximate rdM_{s_d} by roM_{s_d} which further approximates (6) to

$$\forall(z_o, z_d) \in zodM : \forall(s_o, s_d) \in (S_{z_o} \times S_{z_d}) : sodM_{s_o,s_d} \approx roM_{s_o} \times zod_{z_o,z_d} \times roM_{s_d}. \tag{7}$$

In conclusion, using (7), our double approximative Mapper procedure derives a station to station OD-matrix from the zone to zone one. Certainly passenger flows between stations that are important within their respective zones will still be assigned an important part of the traffic between these respective zones.

4.4 Results

In Belgium, we define 497 zones, but only 21 contain more than one station. The number of stations per zone varies from 1 to 7, with the exception of Brussels zone, which contains 31 stations. The zone OD-matrix zod has 19087 zone to zone OD-pairs. Our Mapper, programmed in C++, expands these to 61725 station OD-pairs. We say that the matrix is more *diffused* now. This process takes only 0.97 seconds on a 2.88 GHz dual core Apple machine. Other processes in FAPESP will be much slower than this, so it is not necessary to further speed this up.

4.5 Station Resolution versus Station Countings Recommendations

We have shown that due to the absence of accurate station to station flow data, even a procedure trying to reconstruct it with other data available, can only be an approximation of the original. So, as conclusion of Section 4 we can recommend a more ideal, yet low cost improvement to the data collection procedures for the train operator company.

At ticket purchase time, let the train passengers specify his or her morning source station and morning destination station instead of just recording zones, and keep accurate record of this. This would represent more correctly, the resolution of passenger flow data, which is important to avoid any station within zone uncertainty for the optimization stage. This then, would make the Mapper totally redundant, which is a good thing.

If the OD-matrix is only available on the zone level, we need to rely on countings to obtain the OD-matrix on the station level to do the mapper transformation. The countings should be performed well, which means that:

- Departing passengers should be counted at origin station and arriving passengers at destination station. As demonstrated these countings poM_{s_o} and pdM_{s_d} are both necessary.
- Better still, would be to also ask the departing passenger what is his destination station, as such obtaining $sodM_{s_o,s_d}$ directly. This requires talking to instead of just counting passengers which is more involved. Note that to be entirely accurate even transfer passengers should be excluded in these countings, which makes countings even more complex.
- This is another reason why directly defining tickets and subscriptions in terms of stations instead of zones is largely preferable to these compensating counting methods.

5 The Router: Routing Station OD-Pairs to Determine Flows

The Router, or FA module, has two main tasks. It needs to derive a graph from existing trains and add transfers. Then it needs to perform the actual routing over it, for all station OD-pairs, meanwhile accumulating their flows over the route edges.

To realize this, we have to find the routes that each passenger going from origin station s_o to destination station s_d will choose. Passengers will usually choose the shortest or almost shortest route in time. Some bad experience along a slightly shorter route with varying time, for example due to a transfer often missed, may drive the passenger towards a slightly longer but more duration stable route. For simplicity, we currently supposed that

we can select the shortest route. Before doing the actual routing, we start with constructing the graph that routes will be chosen from.

5.1 Graph Derivation

The graph is the FAPESP graph whose elements are defined in Section 2.1. There are two actors in our graph, trains and passengers. We start with trains.

Dwell And Ride Edges From Existing Train Lines
To produce results relevant to todays operations we started from the existing passenger train set. We focus on the morning peak. We want to end up with a train set that can be copied from hour to hour and as such must be representative for every morning peak hour. We first had to make a decision on what trains to select for inclusion in the graph. We chose the trains that run during the hours 6 to 10 as the ones to start from. Then, per couple of opposite trains occurring every hour, we took one representative direction and hour. We noticed that symmetry is not perfectly maintained within each train couple, but we consistently chose the first train in the couple as representative. There are also deviations from periodicity, which meant that for a series of hourly repetitions, we had to choose a certain representative hour. We took the criterium of selecting the hour of the train that used the most capacity. This means it is riding for the longest time. The rationale here is that, if we are able to plan this train, we will also be able to plan a train that uses less capacity. Among the quasi-symmetrically reversed trains, the same capacity-based selection process is used.

This way, we obtained 205 trains over different categories as they exist in Belgium. These trains stop and pass through 1114 points of which 550 are stations and 564 are other reference points. The combinations of these trains with these points gives 11771 vertices and service edges in our graph. Graph edge numbers are given in Table 2. The train categories are given in order of having few to many stops per distance travelled. Note that the P trains, which are *peak trains*, do not have an opposite train during the same hour, indicating the apparent benefit of asymmetry during peak hours.

Table 2: Belgian Passenger Train Graph Main Figures

Train Type	Lines	Service Edges		Potential Transfer Edges to					Total
		Ride	Dwell	**IC**	**IR**	**L**	**CR**	**P**	
IC	50	2294	2244	2897	2205	1338	989	38	7467
IR	41	1390	1349	2159	1431	1181	682	36	5489
L	92	1723	1631	1319	1184	1542	238	47	4330
CR	20	528	508	989	701	237	850	54	2831
P	2	53	51	35	34	45	50	0	164
Total	205	5988	5783	7399	5555	4343	2809	175	20281

Many Potential Transfer Edges
Apart from ride and dwell edges, Table 2 also shows a lot of *potential* transfer edges, 20281 in total. We explain here what they mean and why there are so many.

The railway companies have defined where they want to guarantee a transfer. They also have minimal times for these, which they consider in planning. There are however, a lot of *other, non-planned, potential* transfers, whose duration corresponds to the difference between planned arrival and departure times of a pair of trains in the same station. If this

time difference is practical, say around 2 to 20 minutes, they will be *exploited* by some passengers. We should not ignore these other transfers.

When we are in the process of scheduling, we do not know the relative order of trains yet. We only know minimal durations of ride and dwell actions m_e per edge e. Suppose a train A arrives at time $t_{arr,A,S}$ in station S and a train B departs at time $t_{dep,B,S}$ in the same station. We model a vertex $v_{arr,A,S}$ for the arrival of train A in S and $v_{dep,B,S}$ for the departure of B in S. We want to model a transfer edge e_{tr} from $v_{arr,A,S}$ to $v_{dep,B,S}$. Suppose as well that a walking time between the platforms of train A and B takes time $m_{e_{tr}}$ The condition that the transfer can be realized on time, is that

$$t_{arr,A,S} + m_{e_{tr}} \leq t_{B,dep,S}. \tag{8}$$

The minimum time $m_{e_{tr}}$ is a known constant but $t_{arr,A,S}$ and $t_{dep,B,S}$ are still unknown. So how can we avoid mapping passengers on a transfer that may go back in time? The answer is that the transfer is always possible, since due to periodicity, from A, there is always a next-hour-B to transfer to, however, sometimes with a high transfer time.

This means that we can and should define potential transfers between all train pairs stopping at a common station, even in both transfer directions. Refer to Figure 1 for the 8 transfers edges generated by the crossing of two train pairs in a station. We have modeled 169 morning trains of which quite some pairs can run along and stop at the same stations, and the number of transfer edges is quadratic in the number of trains per common stopping station. So it is not surprising that 10879 potential transfers can result. Table 2 shows these 10879 potential transfers, divided up between each pair of train categories. Their importance shows the potential interactivity between different categories.

Some reduction in this high potential transfer number could be made here. In order of increasing crudeness, we could disallow transfers

- from a train to other trains that simply go back to where a passenger just came from,
- between trains with the same direction on a common line in every common station. (Selecting a first or last transfer over their common trajects is sensible.)
- in very small stations which are supposedly only end stations or departing stations for people.

For now we try to reduce development time, and still leave all transfer edges in the graph. We prefer to spend our efforts on tuning the retiming MILP model, such that the solver can cope with the full graph in a reasonable time and full optimality can be reached.

5.2 Flow Derivation on a Timed versus Untimed Schedule

To derive how flows are distributed over a graph, we need to distinguish three cases. First, we can use this procedure for an already planned schedule, that has already a duration for each of its edges. We call this a *timed* schedule. Second, in the case of optimization from scratch, no edge durations are known for the initial schedule. We call this an *untimed* schedule. Third, in all next iterations retiming has at least been performed once, so these are called *retimed* schedules. We examine the differences in treatment here.

Flow Derivation on a (Re)Timed Schedule

In both a timed and a retimed schedule, we make use of the planned edge durations d_e. This is in general the minimal time required for the action e, called m_e, plus a certain supplement, called s_e, which is defined in the planning or retime phase respectively. Indeed the task of the planning as well as retime phase, can be said to be defining, for every e a supplement $s_e \geq 0$, on top of the minimal action times m_e. The result is

$$\forall e \in (E \setminus E_{tr}) : d_e = m_e + s_e \geq m_e. \tag{9}$$

So in both contexts, we can use already chosen d_e values. We excluded the transfer edges from 9 because they require special treatment. Their durations can be derived from the other edges as

$$\forall e \in E_{tr} : d_e = (t_{e_+} - t_{e_-} + T) \bmod T, \tag{10}$$

where e_+ is the end vertex of the edge e and e_- its beginning vertex and T is the period of our graph. Since we suppose periodicity T, say one hour, of all trains in our graph, also the transfer between every train pair can occur every hour. Consequently, these two trains can also never be more than one hour apart, so $\forall e \in T_{tr} : d_e < T$. Note that we could add a penalty to (10), being the time for the inconvenience of having to take a transfer instead of spending an equal total time on a slower train without a transfer. A subjective time value is possible here. This has been described by Vansteenwegen and Dewilde et al. [4, 17, 18].

Flow Derivation on an Untimed Schedule

We mentioned before that we can also use this procedure on a schedule, which is still being optimized. This means that in the FAPESP context, it has not passed any retime phase yet. We should take care that, in this case, no prejudice is present in our graphs edge durations. We believe, this means we can choose the values $d_e = m_e$, since this is equivalent with supplements $s_e = 0$, so

$$\forall e \in (E \setminus E_{tr}) : d_e = m_e. \tag{11}$$

This, at least a priori, gives no preference for any edge to get extra slack over other edges.

We again excluded transfer edges from (11). We could, here as well, use (10) to derive transfer edges durations. However, since current edge durations from (11) are not really to be trusted yet, we do not want to derive them from this still uncertain information.

We want to select initial times for the transfers that are not given preference to any of them, so they all need to get the same initial value, say C_{tr}.

$$\forall e \in E_{tr} : d_e = C_{tr} < T. \tag{12}$$

On the other hand, since we need to model peoples choices in choosing train routes, we want to avoid choosing too many transfers. Hence we choose a value for C_{tr} that is equal to the alternative cost in time of being on riding or dwell edges. We took

$$C_{tr} = 15min \tag{13}$$

as an estimate for this penalty. So 15 minutes is considered a balance between a transfer short enough to be considered interesting and taken by passengers and long enough to not

result in people changing forth and back at every station between parallel trains in wanting to skip dwell times longer than this transfer time.

It may seem strange that (12) breaks the usual edge constraints in (10) which seem to have universal validity in any graph where vertices represent absolute times and edges durations between them. In fact, it should, but consider that this is only a postcondition of the retime phase and not a precondition of the reflow phase. Dijkstra and other routing algorithms are certainly able to cope with a schedule satisfying (12) instead of (10), since they only use edge duration variables and no vertex variables.

In conclusion, in the FAPESP iterative context, (11) and (12) are the ones to use for *bootstrapping* the procedure in the initial iteration. All next iterations will be able to use (9) and (10).

5.3 Router Algorithms

The main part of our reflow phase is the router algorithm. There are many algorithms described in literature. We start with a classic shortest path algorithm.

Dijkstra Shortest Path Algorithm

The single pair Dijkstra shortest path algorithm [3, 5] finds a sequence of edges, called a *route*, in a directed graph between two specified vertices v_o and v_d, that has minimal length. The route length is defined as the sum of the length of all edges belonging to this route. In our case, we use as edge length for edge e, the time d_e, so this results in a route with minimum duration. The Dijkstra algorithm only works for non-negative edge lengths. Since our durations are non-negative, this is fine. It uses a greedy breadth first search and a process of relaxation that allows it to make shortcuts. The total running time of this standard algorithm is $O(|V|^2 + |E|)$ which is $O(|V|^2)$ [3]. Our average Dijkstra execution time was 1.28 seconds per pair, on a 2.88 GHz dual core Apple machine, which still seems quite high.

Modified Dijkstra Shortest Path Algorithm

Since for our graph $|V| = 11771$ and $|E| = 11771 + 20281 = 32052$, it is sparse. For sparse graphs, it is recommended [3] to implement a priority queue, with a binary heap, which then results in the *modified* Dijkstra algorithm. The total running time then becomes $O(|V| + |E|lg|V|)$ which is $O(|E|lg|V|)$ [3]. This predicts an asymptotic time saving factor for going to the modified version, of $O(|V|^2/|E|lg|V|) = 40107$. Running the serial Dijkstra shortest path algorithm sequentially over the 61725 station to station pairs, on the same machine, now takes 23 minutes 45 seconds, so 1425s. This is an average of $1425s/61725 = 23ms$ per OD-pair. So the binary heap implementation of the priority queue gained us a factor $1.28s/23ms = 55.6$ which is quite a bit lower than the predicted magnitude for the asymptotic case but still satisfying.

Further Possible Improvements

Implementing the priority queue as a Fibonacci heap instead of a binary heap could still lower execution time to $O(|V|lg|V| + |E|)$ [3].

On multicore systems a parallel version which can execute multiple route findings in separate threads simultaneously could also be implemented.

More essentially, instead of considering all the $O(|V|^2)$ OD-pairs separately, we could cache results of sub-routes already calculated in executions of previous OD-pairs and reuse

them. This idea occurs also in the approach in the *all pairs* Dijkstra Algorithm [3]. Johnson's algorithm has the best asymptotic performance from the ones mentioned by Cormen et al. [3]. Instead of executing singe pair algorithm $|V|^2$ times sequentially which would give a performance of $O(|V|^3+|V|^2|E|lg|V|)$, Johnsons algorithm gives $O(|V|^2lg|V|+|V||E|)$ when using the Fibonacci heap. This looks promising, but since we expect retiming to take much more execution time than the current 23 minutes for reflowing, we, for now, prefer to spend our efforts there rather than here.

We have now described the reflow phase. We now want to execute the reflow phase on the current schedule to know current flows. Then we want to see if we can do better by running the reflow phase on the untimed schedule.

5.4 Reflowing the Current Timed Belgian Schedule

The current schedule is a timed one as defined in 5.2.

Output Data Consistency Check
The first post condition we check is that flows add up in every vertex. We have the *flow law*

$$\forall v \in V \setminus (V_i \cup V_o) : \sum_{e \in v_+} f_e - \sum_{e \in v_-} f_e = 0, \tag{14}$$

where v_+ is the set of in edges of vertex v and v_- its set of out edges. This check passes, which means that flows for each route have been properly accumulated over their edges.

Output Data versus Expectation Checks
We listed all edges sorted by decreasing flows and see if we get results that we recognize in reality. The 10 edges with the highest calculated flows in the country are all around Brussels, which is known as the bottleneck in Belgium. Indeed, a lot of people commute by train to Brussels every day. Edge flows on them vary around 10000 people per day. These are all ride and dwell edges, some consecutive.

The transfer edge with the highest flow is number 866 in the list with 3029 transfer passengers per day, also in Brussels, between a fast IC train and a more local CR train. This means people come from far and have to take a second local train in Brussels to get to their jobs.

We can also look at stations. A first check is to compare the in and out numbers with the ones we derived in the mapper. This is a close match. Where we see differences, they can partly be attributed to the approximations described in the Mapper Section but also to the fact that the absolute numbers we have are averages over the years 2003 to 2007 while our OD matrix dates back from 2002. We see a growth over the years of in and out flows, which varies a bit over different cities.

We checked the transfer flow numbers, per station. From an Infrabel expert, we know what are stations with many transfers in reality. For the current schedule, to see if we obtain the same list, we sum all transfer flows per station and order the stations by total transfer flow in Figure 3. The stations in the list are confirmed to be the *transfer stations*.

For validation of our results for separate flows per train or train pair, we tried to obtain data about passenger counts per ride or transfer edge, but could not get this data.

Further removed from our input data, and largely dependent on the Dijkstra algorithm are the dwell edges. Dwell edges are never counted, because it is not practical because of

Table 3: Biggest Stations: Current, Robust Schedule Flows and Average Edge Durations

Station Name	In= Out	Ride		Dwell		Transfer	
		F	$\overline{D}$	F	$\overline{D}$	F	$\overline{D}$
	p/day	p/day	min	p/day	min	p/day	min
BRUSSEL-NOORD	20202	304820	1.6	158844	0.8	18001	6.9
BRUSSEL-ZUID	24403	321084	1.6	174662	0.8	13715	5.4
GENT-SINT-PIETERS	18201	73259	2.7	10252	2.2	8211	8.2
OTTIGNIES	2929	52873	1.8	17343	0.9	7766	8.1
MECHELEN	8294	95020	2.0	52031	0.4	6949	6.4
DENDERLEEUW	2900	68479	2.5	23959	1.8	7381	6.7
BRUSSEL-CENTRAAL	36983	203596	1.3	59419	1.0	5419	6.3
LEUVEN	10879	80074	2.5	31865	1.2	4313	7.4
LIEGE	6554	31750	2.6	12478	0.6	3521	7.4
NAMUR	9083	43395	2.9	15861	0.6	2950	10.4
ANTWERPEN-BERCHEM	6351	52855	0.8	17751	0.8	2327	6.7
BRUGGE	8590	84921	2.0	38947	0.2	1722	10.2
CHARLEROI	5149	17968	2.3	2601	1.4	1683	10.4
ANTWERPEN-CENTRAAL	8186	24523	1.8	2430	1.0	1647	6.1
KORTRIJK	4500	25185	1.6	7020	0.2	1078	17.0
MONS	4646	18828	3.6	4050	1.4	718	14.0

moving people, so this data is entirely new. For Brussels Central we come to 58257 daily dwell passengers, dwelling 1 minute on average.

We consider our output data starting from the current schedule, validated enough to have confidence in our reflow procedure. We now want to improve on this schedule, by starting from a clean slate, meaning on the untimed, unbiased schedule as defined in 5.2.

5.5 Reflowing the Untimed Belgian Schedule

We also ran our Router on the untimed Belgian schedule with the same trains, but with edge durations $d_e = m_e$ and transfers equal to 15 minutes. This gives the result in Table 4. Of course as for the current schedule reflowing in 5.4 we checked and passed the flow law. However, since the untimed schedule is an abstract one, contrary to the current schedule reflow results from 5.4, we cannot compare with any data from reality.

We see that the untimed schedule numbers in Table 4 are different from timed schedule numbers in Table 3. Comparing both is treacherous though. We have to realize that the untimed schedule does not meet the requirements (9) and (10) for a final, timed schedule yet. The requirement (11) it does meet, results in lower edge durations and averaged per station durations $\overline{D}$ than when it has to meet (9) with $s_e > 0$ for some edges e.

The requirement (12) with $C_{tr} = 15min$ it meets, allows it to optimize flows more transfer time unbiased and hopefully better than when it has to meet (10) directly. Indeed for the timed schedule a couple of opposite transfers will sum to the period T, one hour here. For the untimed one, each transfer takes 15 minutes, so the sum of each opposite transfer couple is only 30 minutes. To be able to compare the two tables, we need to pass the untimed schedule through the retime phase first, so that it will meet (9) and (10). Apart from this retiming, we need to realize that the current schedule is planned to be robust too, while our untimed schedule, even if passed through retiming, will still be *tight* [8], meaning that it has only the minimal amount of supplements $s_e \geq 0$ to make the schedule feasible. So, in general, more than these supplements will have to be included to make it also robust, which is amongst others, needed for *reliable connections* [8]. This can be done by including

Table 4: Biggest Stations: Untimed, Tight Schedule Flows and Average Edge Durations

Station Name	In= Out	Ride		Dwell		Transfer	
		F	$\overline{D}$	F	$\overline{D}$	F	$\overline{D}$
	p/day	p/day	min	p/day	min	p/day	min
BRUSSEL-NOORD	20202	287067	1.6	153069	0.8	11253	15.0
BRUSSEL-ZUID	24403	312473	1.6	169762	0.8	12660	15.0
GENT-SINT-PIETERS	18201	71557	2.7	10480	2.1	7133	15.0
OTTIGNIES	2929	52399	1.8	19399	0.9	5450	15.0
MECHELEN	8294	96507	2.0	54846	0.4	5364	15.0
DENDERLEEUW	2900	64645	2.5	21538	1.4	7885	15.0
BRUSSEL-CENTRAAL	36983	198001	1.3	61262	1.0	779	15.0
LEUVEN	10879	78962	2.5	32845	1.4	2659	15.0
LIEGE	6554	30965	2.6	12370	0.6	2951	15.0
NAMUR	9083	43755	2.9	15663	0.5	3323	15.0
ANTWERPEN-BERCHEM	6351	50715	0.8	16269	0.8	2739	15.0
BRUGGE	8590	82429	2.0	36973	0.2	1511	15.0
CHARLEROI	5149	17541	2.3	1619	1.1	2445	15.0
ANTWERPEN-CENTRAAL	8186	22454	1.8	2573	1.0	470	15.0
KORTRIJK	4500	25976	1.6	7180	0.2	1313	15.0
MONS	4646	18481	3.6	3838	1.5	757	15.0

robustness elements in the goal function [4, 17, 18]. So, under the conditions that retiming is performed and robustness measures are taken, direct comparison with the current schedule will be fair.

5.6 Route Registration versus Edge Countings Recommendation

In Section 5, we tried to reconstruct the intentional route of all train passengers. This is something we have to do, because we lack this initial information.

- But, we recommend that, at ticket purchase time the train operator company asks the traveller to supply them with the route he would normally choose himself, including specific trains and possible transfers between them. To ease this question, some routes could be suggested and the traveller could get these as a multiple choice list, including an *other route* choice which he then mentions himself. These recorded routes could be used in our system as initial routes, determining initial flows. This is the best possible initial route information we could get. This does not make the reflow phase redundant. Reflow will still be necessary for the generation of routes at ticket purchase time, and for the mentioned reflow iterations.

- As mentioned in 5.4, we would have liked to be able to check our output data with passenger numbers counted in real life, but could not obtain this data. So we can recommend to the railway companies, that, in addition to the low cost and essential questions to be asked at ticket purchase time, counting passenger flows in trains helps to complete or update the changing passenger flow picture. Since at dwell time, passengers are moving through a train a lot, it is easier to count passengers during riding and even counting transfer passengers between two trains. By making the differences, using the flow law (14) at arrival or/and departure time, the number of dwell passengers can then be derived.

6 FAPESP Extension: CODFAPESP

Dependent Iterations
In flow allocation in FAPESP, we model the expected reaction of passengers potentially changing train routes between their constant origin destination pairs, when the timing of their schedule is changed by the railway company.

What we have not modeled yet, is that, when the railway company changed the timing of the schedule, people can also have the following reactions

- People who are not currently using the train, but other means of transport like their car or a bus, are suddenly attracted by a new shorter and robust train route from home to their daily job location and back. They could decide to buy a subscription and become new members of the zone to zone origin destination matrix.
- Other people may be on a route that due to low f_e values is less important to the rail company. As a consequence, service on this route may go down, reflected in higher d_e values along this route. Some of these people may give up using the train and switch to other means of transport.

We call these people-reactions reflected in a Change of the OD-matrix the COD phase or reOD phase. Obviously, decisions about buying or giving up a railway subscription happen over a longer time, probably months or even years, than the usually, daily decisions about which route to take to and from work. We have *little hope* that modeling the reOD phase can be usefully accurate. Indeed it is much harder to model since more factors play a role in this human decision than purely calculating the shortest duration path to work. Luckily, there is also *little need* of modeling this process, precisely because changes in od_{z_o,z_d} values happen much slower than changes in f_e values. We take the simple wait-and-see approach, where we let the passengers decide for themselves and seasonly, use the result of adapted ticket sales in our model.

This rationale leads to an adapted process with an extra longer term loop for people-decided OD-matrix adaptations. We call this problem the CODFAPESP, for Flow Allocation and Periodic Event Scheduling under Changing OD matrix Problem. The process flow chart to tackle this problem is given in Figure 3.

Note that the CODFAPESP process has two nested loops with different execution frequencies as well as contexts. The outer loop describes that every season, we will need to propagate the changes in ticket sales in our model. The inner loop needs to be executed at planning time, until converged to a satisfying solution. Only then can the new planning be put in place in reality. We do not want passengers and the railway companies having to perform this inner loop in reality, implying too slowly or even never converging three-month iterations, which is what happens now.

7 Conclusions

This paper presents three major contributions. Firstly, it builds up the theoretical extension of the PESP retime phase with the passenger reflow phase to the FAPESP. Secondly a practical implementation of the FAPESP is demonstrated, which is not so straightforward as the theory suggests. Thirdly, we give some recommendations to the railway companies about data they can and should gather when they want to focus more on passenger service and optimize the timetable from a passengers point of view.

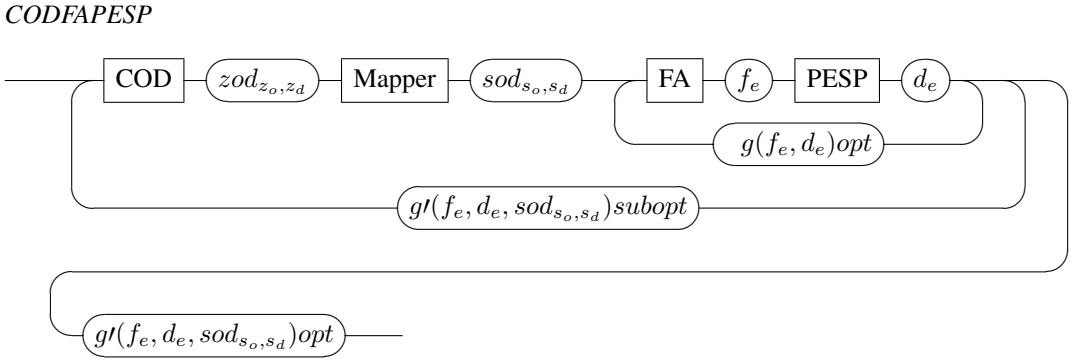

Figure 3: CODFAPESP Two Leveled, Two Phase Flow Chart
Showing effect of long term change in demand

In order to optimize the timetable later, the procedure described in this paper allows to determine passenger flows for a railway network. The passenger flows, together with the action durations, are the variable components in our minimized goal function of total passenger travel time. This initiates a practical process for railway timetable optimization.

Quite apart from the goal of timetable optimization purposes, the reflow phase can also be used separately to derive passengers flows for a complete schedule. This has been done for the Belgian network for the schedule in operation and resulted in accurate passenger flow numbers. This information has never been available before and can now be used by the Belgian railway infrastructure company, Infrabel, to weigh choices and take decisions with.

8 Further Work

We identified some further work. We need to further verify the result of flow allocation phase with future, partial passenger counting data as it becomes available from the railway company. We can still use more refined routing algorithms that account for passengers choice spread amongst all the best routes instead of only an arbitrary best route now. We can also use more balanced routing algorithms that account for passengers choice spread amongst good routes instead of only the best route or best routes. We want to complete the retime phase so that the reflow and retime phase can be iterated over and convergence can be checked. We also would like to perform FA and PESP in a single phase [14], for its own right and to be able to compare its speed and results to our two phase iterative method.

References

[1] Bouma, A., Oltrogge, C., "Linienplanung und Simulation für öffentliche Verkehrswege in Praxis und Theorie", *ETR*, Nr 43, H. 6, pp. 369-378, 1994.

[2] Cordone, R., Redaelli, F. "Optimizing the demand captured by a railway system with regular timetable", *Transportation Research Part B: Methodological*, 2010.

[3] Cormen. T. H., Leiserson, C. E., Rivest, R. L., "Introduction to Algorithms", *The MIT Press, Mc Graw Hill*, 1989

[4] Dewilde, T., Sels, P., Cattrysse, D., Vansteenwegen, P., "Defining Robustness of a Railway Timetable", *Proceedings of RailRome*, under review, 2011.

[5] Dijkstra, E. W., "A Note on Two Problems in Connection with Graphs." *Numerical Mathematics*, Vol. 1, pp. 269-271, 1959.

[6] Exel, M., Velzeboer, J., Warmerdam, J., Colloquium Vervoersplanologisch Speurwerk 2001: "Tariefdifferentiatie in treintoedelingsmodel TRANS" *CVS 2001*, deel 3, pp. 1483-1498, 2001.

[7] Goverde, R.M.P., "Synchronization Control of Scheduled Train Services to Minimize Passenger Waiting Times", *Proceedings of the 4th TRAIL Annual Congress*, part 2, TRAIL Research School, Delft, 1998.

[8] Goverde, R.M.P., "Improving Punctuality and Transfer Reliability by Railway Timetable Optimization", *Proceedings of the 5th TRAIL Annual Congress*, TRAIL Research School, Delft, 1999.

[9] Hofker, H., Suurland, M., Warmerdam, J., Colloquium Vervoersplanologisch Speurwerk 2000: "Wie betaalt, bepaalt!" *CVS 2000*, part 3, pp. 1547-1560, 2000.

[10] Kroon, L., Dekker, R., Vromans, M.J.C.M., "Cyclic Railway Timetabling: A stochastic Optimization Approach", Geraets, F., Kroon, L., Schöbel, A., Wagner, D., Zaroliagis, C.D. (eds), *Algorithmic Methods for Railway Optimization. Lecture Notes in Computer Science*, pp. 41-66, 2007.

[11] Liebchen, C., "Symmetry for Periodic Railway Timetables", *Electronic Notes in Theoretical Computer Science* Vol. 92, pp. 34-51, 2004.

[12] Nachtigall, K., "Periodic network optimization with different arc frequencies", *Discrete Applied Mathematics*, Vol. 69, pp. 1-17, 1996.

[13] Schöbel, A., Knödl, H., "Trendanalyse von Verspätungsentwickelungen im Personenfernverkehr", *ETR*, Nr 11, November 2006, pp. 806-809, 2006.

[14] Schmidt, M., Schöbel, A., "The Complexity of Integrating Routing Decisions in Public Transportation Models", *10th Workshop on Algorithmic Approaches for Transportation Modeling Optimization and Systems, ATMOS'10*, pp. 156-169, 2010.

[15] Schrijver, A., Steenbeek, A., "Spoorwegdienstregelingontwikkeling (Timetable Construction)" *Technical Report, CWI, Amsterdam*, (in Dutch), 1993.

[16] Serafini, P., Ukovich, W., "A Mathematical Model for Periodic Scheduling Problems", *SIAM Journal on Discrete Mathematics*, Vol. 2, pp. 550-581, 1989.

[17] Vansteenwegen, P., Van Oudheusden, D., "Developing railway timetables which guarantee a better service", *European Journal of Operational Research (EJOR)"*, Vol. 173, pp. 337-350, 2006.

[18] Vansteenwegen, P., Van Oudheusden, D., "Decreasing the passenger waiting time for an intercity rail network", *Transportation Research Part B: Methodological*, Vol. 41, pp. 478-492, 2007.