

**Solving the train scheduling problem in a main station
area via a resource constrained space-time integer
multi-commodity flow**

Martin Fuchsberger

Master Thesis

· Zurich, winter term 2006/2007 ·

Institute for Operations Research
ETH Zurich

Professor: Prof. Dr. Hans-Jakob Lüthi

Supervisors: Gabrio Curzio Caimi and Dr. Fabian Chudak

Abstract

We address the problems of generating conflict-free train routings and train schedules in main station areas. As work in this field has been done, two existing models are presented. Then an improved model based on an integer multi-commodity flow formulation incorporating railway topology, passing times, and speed profiles of trains is proposed. We algorithmically generate substantially stronger constraints than in the previously used approaches, by considering for each resource all trains simultaneously (instead of just two). Solving the resulting integer linear program with real data, it is possible to find feasible solutions within seconds even for cases when previous approaches fail. It may be possible that not all trains can be scheduled for certain problem instances and hence the problem instances are infeasible. Instead of just trying to solve the feasibility problem, we introduce an objective function which tries to schedule as many trains as possible.

Contents

Acknowledgments	vi
1 Introduction	1
1.1 Background	1
1.2 Restriction to main station areas	1
1.3 Problem Statement	3
1.4 Related Work	3
1.5 Outline	3
2 Problem Formulation	4
2.1 Train Scheduling	4
2.2 Pulsed Train Scheduling	6
2.3 Pulsed Train Scheduling with favored departure times	7
2.4 Train Routing	7
3 Train Routing Models	8
3.1 Conflict Graph	8
3.2 Tree Conflict Graph	9
3.3 Resource Tree Conflict Graph	12
3.3.1 Allocation of a Resource	12
3.3.2 Improving Conflict Modeling for a Resource	16
3.3.3 Resource Tree Conflict Graph Model	21
4 Model for Pulsed Train Scheduling with favored start times	23

5	Results for Train Routing Problem	25
5.1	Conflict Graph and Tree Conflict Graph	25
5.2	Resource Tree Conflict Graph	26
6	Results for Pulsed Train Scheduling Problem	27
7	Conclusion and Outlook	30
	Bibliography	30

List of Figures

1.1	Swiss railway network and a restriction to the main station area Bern, Switzerland	1
1.2	Station region of Bern	2
1.3	Track switches in a main station area	2
2.1	Double node vertex graph	4
3.1	Conflict graph and colored independent set	8
3.2	Example of a track topology	9
3.3	Routing Trees based on the example track topology	9
3.4	Conflict in the track topology	10
3.5	Routing Trees with a conflict edge	10
3.6	Conflict graph with highlighted conflicts	10
3.7	Tree Conflict Graph and Multi Commodity Flow	11
3.8	Typical resources occurring in a main station area.	13
3.9	Signals (stop)	14
3.10	Signals (90 km/h)	14
3.11	Signals (Clear line)	14
3.12	Allocation of a resource by several train routes	16
3.13	Improved Conflict Modeling	16
3.14	Algorithm 3: Sort start- and end-times	17
3.15	Algorithm 3: Look at first end time A_2 . The Start times A_1 and B_1 are gathered as a conflict clique.	18
3.16	Algorithm 3: The interval A has ended, remove A from the list.	18

3.17	Algorithm 3: Look at the second end time $B2$. No new start time was inserted; Nothing to do.	18
3.18	Algorithm 3: The interval B has ended, remove B from the list.	19
3.19	Algorithm 3: Look at third end time $D2$. The Start times $C1, D1, E1$ and $F1$ are gathered as a conflict clique.	19
3.20	Algorithm 3: Final end time has been reached. All conflict cliques have been found.	20
3.21	Finding Cliques: Special case	20
3.22	Resource tree conflict graph	21
4.1	Pulsed departure times - Part 1	23
4.2	Pulsed departure times - Part 2	23
6.1	Pulsed Train Scheduling (Bern East 2003): Number of nodes	28
6.2	Pulsed Train Scheduling (Bern East 2003): Number of conflict cliques	28
6.3	Pulsed Train Scheduling (Bern East 2003): Average clique size	29
6.4	Pulsed Train Scheduling (Bern East 2003): Computation times	29

List of Algorithms

1	Calculation of the occupation time interval	13
2	Calculation of the minimal braking time interval	15
3	Finding conflicting cliques of intervals from a set of intervals	17

Acknowledgments

I wish to express my sincere appreciation to all people who contributed to this master thesis in one way or another. Especially, I want to thank Prof. Dr. Hans-Jakob Lüthi for giving me the chance to work on this topic. I owe my gratitude to the flawless supervision of Gabrio Curzio Caimi and Dr. Fabian Chudak. For many helpful comments, discussions and support, I am indebted to Vania Eleuterio, Michael Guarisco, Marc Reimann and Rico Zenklusen. I would also like to thank the Swiss Federal Railways for providing helpful data for the studied test case.

Furthermore I especially want to thank my family and friends for supporting me. I am indebted to Christine Füllemann for her patience and love.

Chapter 1

Introduction

1.1 Background

Railway traffic in Switzerland has increased considerably during the last few years: According to [SBB05] the Swiss Federal Railways (SBB) transported in year 2005 about 250 million passengers and 60 million tonnes of freight. Moreover, railroad transportation will further grow for both passenger and freight transportation. These developments increase the demand for denser timetables. When increasing the density of the timetable, scheduling trains becomes more and more difficult. The reasons are a lot of restrictions and limits. Examples for these factors are the track topology, rolling stock, human resources and service requirements. An automatic generation of conflict-free timetables in reasonable time can be very helpful in order to evaluate several alternative timetables. Therefore, the interest in automatically generating railways timetables has increased over the past years.

1.2 Restriction to main station areas

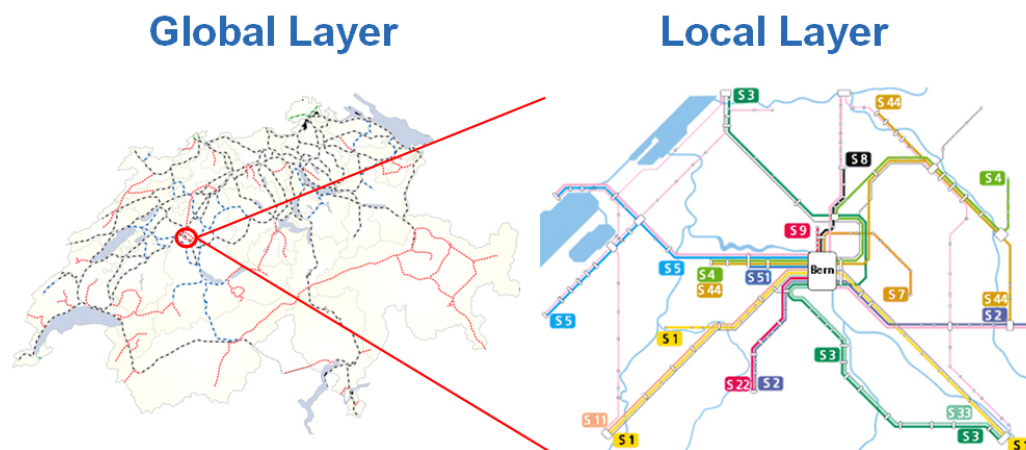


Figure 1.1: Swiss railway network and a restriction to the main station area Bern, Switzerland

In a railway network the main station areas are the bottlenecks ([ZKR⁺96], [Her05]). Different lines converge in proximity of main stations and the track topology can get very

dense and complex (See Figure 1.3). Furthermore the traffic is concentrated in this areas, since almost all trains have to drive through the main station and additionally they have to slow down or even stop. In this master thesis we focused on single main station areas (See Figure 1.1).

A typical main station area in Switzerland is Bern (See Figure 1.2).

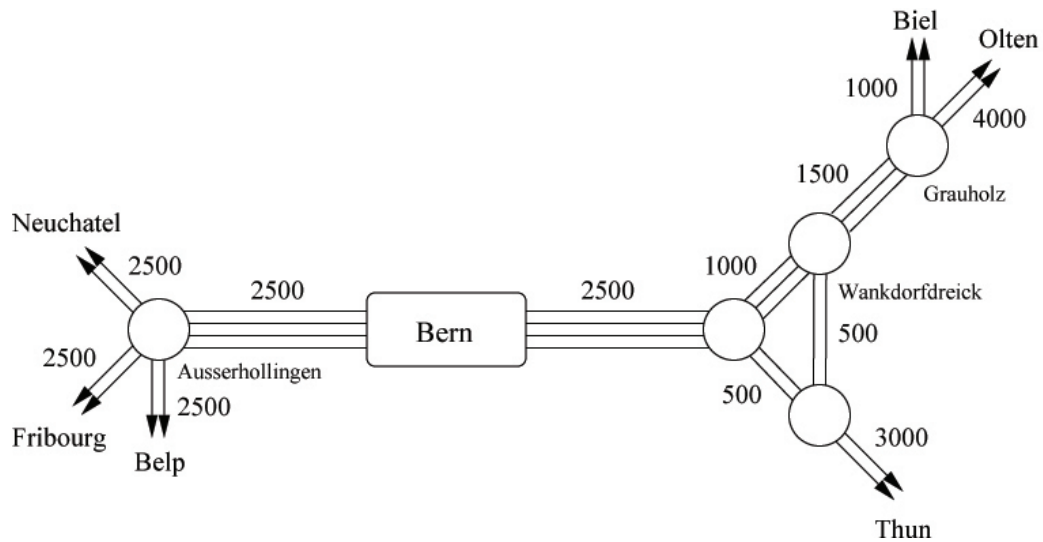


Figure 1.2: Sketch of the station region of Bern with approximate distances given in meters. The circles represent track switches which allow for crossing in this areas.

This area covers a radius of about 6 km. A total of six directions are connected through roughly 500 track switches and sections with the main station. These track switches enable the trains to cross rail tracks, which then enable interconnections between the directions (See Figure 1.3).



Figure 1.3: Track switches in a main station area

1.3 Problem Statement

Our research focuses on the conflict-free scheduling of a list of trains in a main station area. In particular, for each train an itinerary through the track topology with passing times at each relevant point has to be determined. Furthermore, safety restrictions, driving behavior of the trains and interconnections have to be considered. By applying efficient and scalable algorithms, we want to find feasible train schedules for the trains. In case that not all the trains of the list can be scheduled, we want to provide some feedback.

1.4 Related Work

Related work (see [HKL05]) reports two principal approaches to the problem of finding a schedule for a whole railway network: one abstracting from the detailed track topology and the other considering partial detailed topologies of the network. For our research we restrict ourselves on main station areas, and hence we list only the efforts in this area. Two different approaches to find conflict-free train schedules for main station areas can be found in ([CC03], [EVS05]). Other work considered checking the feasibility of a given rough timetable ([CHB05], [ZKR⁺96], [Her05]).

1.5 Outline

In Chapter 2 we define the train scheduling problem, so that we can derive the pulsed train scheduling problem and the train routing problem.

We introduce in Chapter 3 three conceptual models for the train routing problem and show in 4 how one of them can be extended to solve the pulsed train scheduling problem. Note, that we do not solve the train scheduling problem itself, but a discretized version (the so-called pulsed train scheduling problem).

Chapter 5 and 6 present results for the train routing problem and the pulsed train scheduling problem based on data of the main station area Bern, Switzerland.

In the last chapter a conclusion and an outlook is given.

Chapter 2

Problem Formulation

In this chapter we want to give a formulation of the different types of problems, that can arise during the creation of a timetable. We start with the most general problem and continue by reducing the degree of freedom and end up with the train routing problem. Chapter 3 describes models and algorithms for solving the train routing problem and in chapter 4 we extend one of the models to solve the pulsed train scheduling problem. Note that we do not present any models for the general train scheduling problem, since we are more interested in the pulsed scheduling problem, which is part of a SBB project called Puls90 project [Roo06].

2.1 Train Scheduling

Before we can define the train scheduling problem, it is necessary to see how a railway network, a service intention and departure time windows are defined:

A railway network (i.e. the network track topology) can be represented by a conventional graph with the exception that each vertex is doubled, i.e. each vertex has a unique partner (figure 2.1):

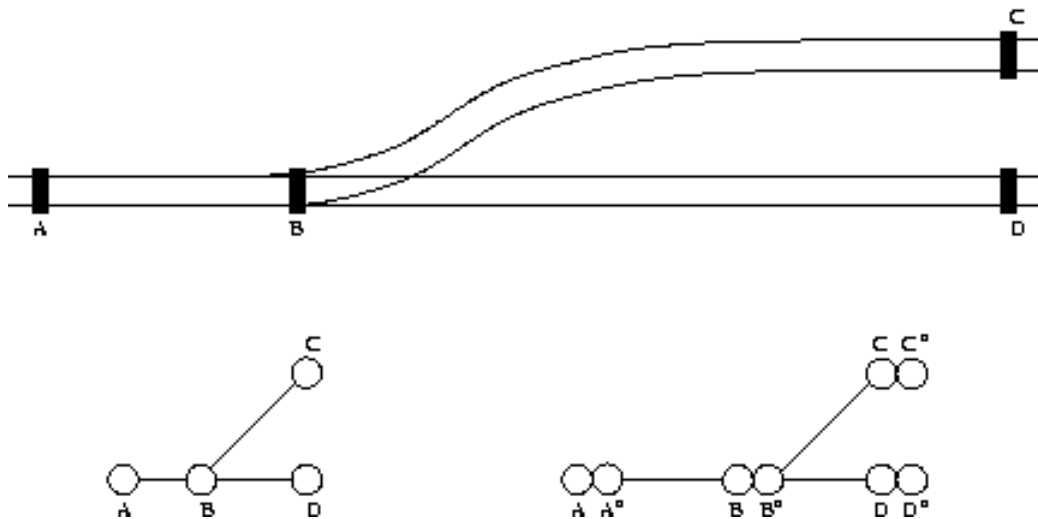


Figure 2.1: An example track network with the transformation to the double node vertex graph.

Formally this is described as follows:

Definition 2.1 (Double Vertex Graph)

Let V be a finite set of vertices, $E \subseteq V \times V$ a finite set of edges between the vertices, i.e. $E = \{(u, v) | u, v \in V\}$. Moreover, let $\circ : V \rightarrow V$ be a mapping, the so called joining mapping, which satisfies $\circ(v) \neq v$ and $\circ(\circ(v)) = v$ for all vertices $v \in V$. Then a triple $D = (V, E, \circ)$ is called a double vertex graph. Write v° for $\circ(v)$

The motivation for the introduction of double vertex graphs (see [Mon92]) is the notion of a path (or route) in the network. There is a simple rule in the double vertex graphs preventing the exploration of impossible paths. This rule is: *Never use more than one outgoing edge of a single vertex—always use both partner vertices to describe a path.* Hence a path in a double vertex graph always follows *node-node-edge-node-node-edge...* Formally a path p is described as follows:

$$p := \{v_1, v_1^\circ, (v_1^\circ, v_2), v_2, v_2^\circ, (v_2^\circ, v_3), \dots\}$$

The representation of a railway network using a double vertex graph is crucial and thus it is assumed that the network topology is always represented as a double vertex graph D . In order to precisely describe the train service intention, some vertices in the double vertex graph receive a special tag. The set of vertices corresponding to platforms is called *platform vertices*—denoted by V^S ; and *portal vertices* V^P representing the border nodes of the network. In order to use the track network a set of lines has to be defined. Lines are described by a set of portal and platform vertices, outlining the route of a train. Moreover, the rolling stock serving a line has to be specified. This information is summarized in the train service intention:

Definition 2.2 (Train Service Intention)

A given train service intention consists of a set of train lines. A single line l_i is described by the following six-tuple:

- the type of the train w_i including the rolling stock
- optionally some pass-through points at specific locations within the station area (e.g. stops at minor stations located in the station area of the main station)
- an incoming portal node $v_i^{Pa} \in V^P$
- a platform node $v_i^S \in V^S$
- an outgoing portal node $v_i^{Pd} \in V^P$
- list of all interconnection possibilities

The first item characterizes the train serving line l_i : w_i contains all information—like the behavior while accelerating or breaking, length, weight—which is needed for further calculations. This information is assumed to be given. The second item stores information whether a train should stop at additional stations within the station area. The three nodes roughly determine the route of the train through the station area. The set of the triples $\{(v_i^{Pa} \in V^P, v_i^S \in V^S, v_i^{Pd} \in V^P)\}$ is called *line plan*. The last item contains the interconnection conditions between the trains.

In order to describe exactly how trains drives through the track network, we introduce the definition of a train route:

Definition 2.3 (Train Route)

A Train Route describes exactly how a train drives through the track network: It is a path in a double vertex graph where at each node v of the path is augmented by the following three items:

- arrival time
- departure time
- velocity

Since several trains drive through the track network safely we define conflict-free train routes:

Definition 2.4 (Conflict-free Train Routes)

A set of train routes T are called conflict-free if and only if each node of the track network is allocated by at most one train route at any time. The allocation of a node of the track network is explained later in subsection 3.3.1.

For train scheduling problems we also consider time windows in which the trains are required to depart at their platform or portal:

Definition 2.5 (Departure Time Window)

A Departure Time Window for a train is a closed time interval in which a train

- entering the main station area has to depart at the incoming portal node $v_i^{Pa} \in V^P$
- leaving the main station area has to depart at the platform node $v_i^S \in V^S$

We assume here, that the departure time windows are given by a nationwide timetable or that they are derived from the customer demands. They are treated as input for the train scheduling problem.

Now all the necessary terms are introduced and we can define the Train Scheduling Problem:

Definition 2.6 (Train Scheduling Problem)

Inputs:

- Track topology of a main station area
- Train service intention
- Departure time windows

Goal: Find a feasible (conflict-free) train schedule.

2.2 Pulsed Train Scheduling

The SBB Puls90 project strives to pulse trains in main stations in periodic intervals [Roo06]. According to this idea the departure time windows of a train scheduling problem can be cut into discrete (pulsed) times, resulting in fixed timings. These timings can then be used to formulate the pulsed train scheduling problem:

Definition 2.7 (Pulsed Train Scheduling Problem)

Inputs:

- Track topology of a main station area
- Train service intention
- Discretized Departure Times according to a pulse time T

Goal: Find feasible (conflict-free) train routes.

2.3 Pulsed Train Scheduling with favored departure times

Some of the possible pulsed departure times of a train are often preferred over others. This is reasonable to assume, since preferred departure times may have reduced logistic complexity at the main station or reduce the scheduling difficulties outside the station. The problem formulation is now slightly different:

Definition 2.8 (Pulsed Train Scheduling Problem with favored departure times)

Inputs:

- *Track topology of a main station area*
- *Train service intention*
- *Discretized departure times according to a pulse time T*
- *Priority for each departure time*

Goal: Find a feasible (conflict-free) train route with the maximum benefit according to the priorities of the departure times.

2.4 Train Routing

We will now look at the train routing problem, since the pulsed train scheduling problem is a composition of train routing problems: If we fix the selectable departure times of a (pulsed) train scheduling problem for each train, we can derive the train routing problem:

Definition 2.9 (Train Routing Problem)

Inputs:

- *Track topology of a main station area*
- *Train service intention*
- *Fixed departure time for each train*

Goal: Find a feasible (conflict-free) train route

A solution to one of the derived train routing problems is implicitly a solution to the original train scheduling problem. This insight provides the idea on how the models for the train routing problem can be extended, such that they can be used to solve the pulsed train scheduling problem.

Chapter 3

Train Routing Models

3.1 Conflict Graph

Zwaneveld et al. introduced in [ZKR⁺96] the conflict graph model. In this model each possible train route through the track topology i.e. an exact itinerary corresponds to a node in the conflict graph. We define a conflict between two routes, if it is not possible to choose both routes at the same time and still fulfill the safety restrictions. If two routes are in conflict, the corresponding nodes are connected by a conflict edge. Additionally, as one train can choose only one route, the nodes of a single train are interconnected and form a clique.

Definition 3.1 (Conflict Graph)

Given a set of routes P and a set of conflicts C a undirected Graph $G = (V, E)$ is called a Conflict Graph of the corresponding track routing problem if

- There is a one-to-one mapping M between P and V ;
- $\forall c = (p, q) \in C : \exists e = (v, w) \in E$ with $M(p) = v, M(q) = w$;
- $\forall p, q \in P$ belonging to the same train: $\exists e = (v, w) \in E$ with $M(p) = v, M(q) = w$.

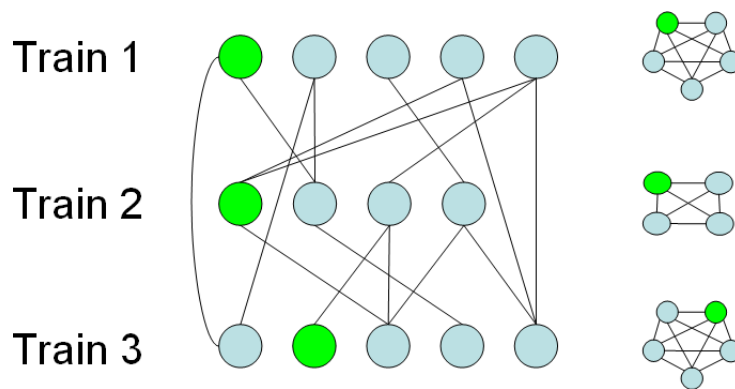


Figure 3.1: Conflict graph and colored independent set

A feasible solution to the train routing problem is equivalent to an independent set in the associated conflict graph with cardinality equal to the number of trains. Zwaneveld et al. made the approach [ZKR⁺96] of solving the problem using an integer linear program:

Definition 3.2 (Conflict Graph - ILP Formulation)

Find x_{ij} such that:

1. $\sum_{j=1}^{m_i} x_{ij} = 1 \quad \forall i = 1, \dots, n$
2. $x_{ij} + x_{kl} \leq 1 \quad (v_{ij}, v_{kl}) \in E, i \neq k$
3. $x_{ij} \in \{0, 1\}$

where x_{ij} indicate whether node v_{ij} is in the set or not and n is the number of trains.

1. allows only one node for a train to be chosen. 2. allows only one of the end-nodes of a constraint edge to be chosen (conflict constraint) and 3. is the restriction to boolean (indicator) variables.

Using this approach and a standard commercial solver, finding exact solutions for larger problem instances takes too much time. Herrmann proposes an heuristic attempt (randomized fixed point iteration, [Her05]), which failed to find solutions for particularly dense instances. Hence, one tried to improve on the structure of the model (see next section).

3.2 Tree Conflict Graph

The conflict graph models conflicts between routes. However it does not tell where exactly in the topology the conflict is located. To bind the conflicts to certain points in the topology, the routes need to be unfolded into chains of topology points. Consider the sample double node vertex graph for the network topology (see figure 3.2).

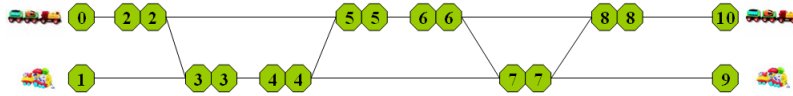


Figure 3.2: Double node vertex graph of a track topology example

There are two trains which should be routed through this network: The first train should drive from node 0 to node 10 and the second train from node 1 to node 9. If the possible routes of the two trains are unfolded as chains and gathered into a tree, we receive two routing trees (see figure 3.3).

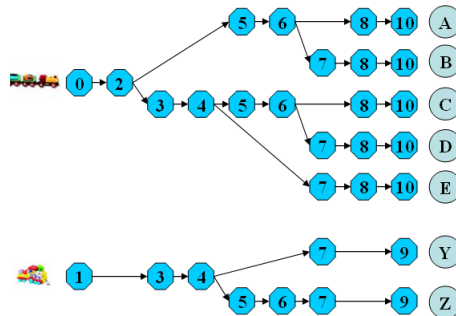


Figure 3.3: The possible routes (A,B,C,D,E,Y,Z) associated with the two trains through the track topology (figure 3.2) is represented as a tree. This tree is the backbone of the tree conflict graph model.

Suppose the routes of the first train (C, D, E) and the route of the second train (Y, Z) are in conflict at the node 3 of the track topology (see figure 3.4).

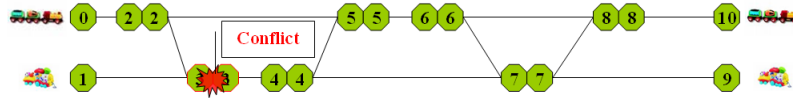


Figure 3.4: The track topology example with routes conflicting at node 3

To include conflicts into the routing tree we add a conflict edge between the nodes of the routes which are in conflict. In our example this is an edge between the two route nodes 3 of the trees (see figure 3.5).

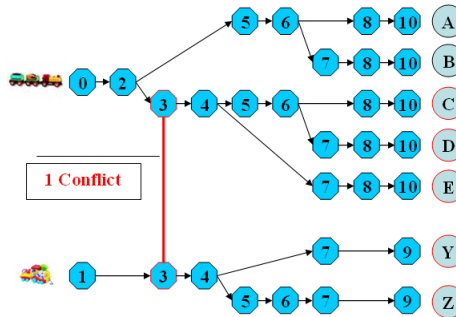


Figure 3.5: The routes C, D, E are in conflict with the routes Y and Z at the topology node 3. Hence, a conflict edge between the route nodes 3 of the trees is added.

The result are interconnected trees, where each tree is associated to one train and the leaves of a tree correspond to the different routes of one train in the previous conflict graph model. The connections between the trees represent conflicts between routes of trains. The tree conflict graph model has a significant advantage: The conflicts between routes can now be bundled into conflicts between subtrees. In figure 3.5 one conflict at the topology point 3 describes the six conflicts between the routes (C, D, E) and routes (Y, Z). In the previous conflict graph model for each of these conflicts a conflict edge had to be built (see figure 3.6).

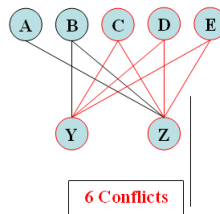


Figure 3.6: The routes C, D, E are in conflict with the routes Y and Z . In the conflict graph each conflict between two routes has to be covered by exactly one edge.

By using the information where the conflicts are located, the number of conflict edges are reduced. Herrmann and Caimi followed this idea in [HC06] and came up with the concept of a *Tree Conflict Graph*:

Definition 3.3 (Tree Conflict Graph)

- For each tuple $(train, time, topology\ element, velocity)$ of a route a vertex is created.
- Conflicts are modeled as (undirected) conflict edges between two tuples (vertices).
- Directed edges model the routes through the track topology from origin to destination.

We are now looking for directed paths from each root node of the trees to one of the leaves, where the directed paths are not interconnected by a conflict edge. We can formulate this search for directed paths as a multi-commodity flow [AMO93] problem. To do that, four steps are necessary:

1. Add sources and sinks
2. Assign variables x_{ij} to each directed (flow) edge
3. Introduce a flow equal to 1 from each source to sink
4. Add a conflict constraint for each conflict edge

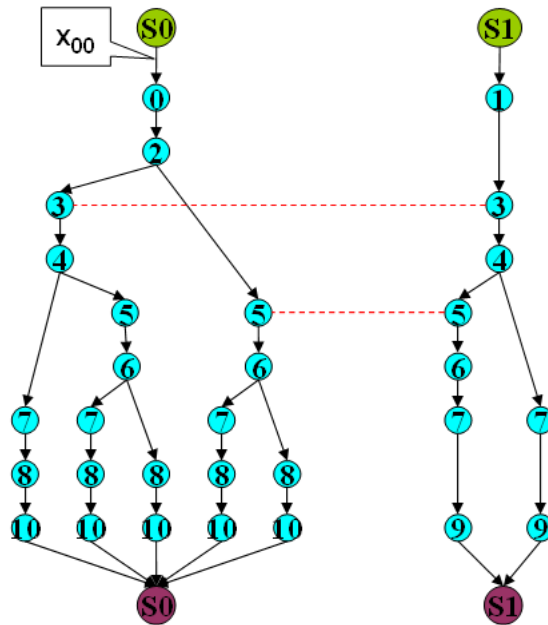


Figure 3.7: Tree Conflict Graph and Multi Commodity Flow

In figure 3.7 you see an example tree conflict graph with the added sinks and sources. The integer linear program follows directly:

Definition 3.4 (Tree Conflict Graph - ILP Formulation)

Find x_{ij} such that:

1. $\sum_{x_{ij} \in S_i} x_{ij} = 1 \quad i = 1 \dots n$
2. $\sum_{x_{ij} \in T_i} x_{ij} = 1 \quad i = 1 \dots n$
3. $x_{ij} = \sum_{m: v_{ij} \rightsquigarrow v_{im}} x_{im} \quad \forall i, j : x_{ij} \notin \bigcup_{i=1}^n T_i$
4. $x_{ij} + x_{lm} \leq 1 \quad \forall i, j, l, m \text{ where node } v_{ij} \not\perp v_{lm}, i \neq l$
5. $x_{ij} \in \{0, 1\}$

where S_k is the set of edges connected to the source node of train k and T_k is the set of edges connected to the sink node of train k . Note, that $v \rightsquigarrow w$ means that node v is ancestor of w , $v \not\perp w$ represents a conflict between node v and node w and n is the number of trains.

Items (1.) - (3.) contain the flow preservation constraints: The inducted flow at each source is equal to one (1.) and the outflow at each sink is equal to 1 (2.). Between the sinks and the sources the flow is preserved (3.), where we do not have to consider the leaf nodes.

Item (4.) states that at most one end-node of a conflict edge may be chosen and item (5.) restrict the variables to boolean values.

Note: Computed results with the tree conflict graph model are presented in Chapter 5.

3.3 Resource Tree Conflict Graph

3.3.1 Allocation of a Resource

Before we define the resource tree conflict graph it is necessary to understand first what we mean exactly by the terms train route in a tree conflict graph, resource and how they are allocated by the train routes:

Definition 3.5 (Train Routes in a tree conflict graph)

A train route in a tree conflict graph for a train is a path from the root of the corresponding tree in the tree conflict graph to one of its leaves.

A train route is calculated in the following way:

1. Calculate from the Double Vertex Graph all the feasible paths.
2. Using the speed profile of a train approximate the velocity, arrival and departure times at the nodes of the calculated paths.

Definition 3.6 (Resource)

A resource is a composition of track elements, such that if more than one train would allocate an arbitrary partition of the track elements at the same time, a conflict is certain to occur.

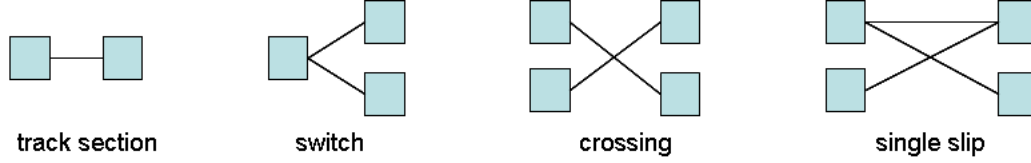


Figure 3.8: Typical resources occurring in a main station area.

In figure 3.8 some typical resources occurring in a main station area are drawn. Each train route allocates several resources for closed time intervals, the allocation time intervals:

Definition 3.7 (Allocation Time Interval)

A Allocation Time Interval is always associated with a resource and a train route. It represents the time during which a train route would allocate the resource. According to [Gra07] it is the composition of

- Occupation time interval
- Minimal braking time interval
- Additional safety related time intervals

Definition 3.8 (Occupation Time Interval)

The occupation time interval of a resource and a train route is the maximum time interval, during which any part of the train would occupy at least one of the track elements contained in the resource.

To calculate the occupation time interval, the departure time at the entering point is subtracted from the departure time at the leaving point. The departure time at the leaving point denotes when the train head is leaving the resource. Hence, we have to approximate the time needed for the train tail to leave the resource as well. This is done by dividing the train length through the minimal possible velocity to completely leave the resource (see Algorithm 1).

Algorithm 1: Calculation of the occupation time interval

Input: entering node n_e , leaving node n_l , train k , resource tree conflict graph G , train route U

Output: occupation time interval start T_s and end T_e

T_s = departure time at node n_e from (U, G) ;

T_e = departure time at node n_l from (U, G) ;

$v_{min} = \infty$;

foreach train route r of train k leaving node n_l **do**

if $v_{min} > \text{velocity of train } k \text{ with route } r \text{ at node } n_l$ **then**

 | $v_{min} = \text{velocity of train } k \text{ with route } r \text{ at node } n_l$;

end

end

$T_e = T_e + \text{round} \left(\frac{\text{train-length}}{v_{min}} \right)$;

return $[T_s, T_e]$;

Definition 3.9 (Minimal Braking Time Interval) *The minimal braking time interval of a resource and a train route is the time interval with minimal length, during which the train would be able to stop before it would occupy a track element of the resource.*

In the swiss railway network there are signals distributed to inform the train drivers on the occupation states of resources (Called Signal System L, See Figures 3.9-3.11). The signals are placed in such a way, that regardless of the actual speed of a train, the train driver would always be able to stop the train before entering the corresponding resource.

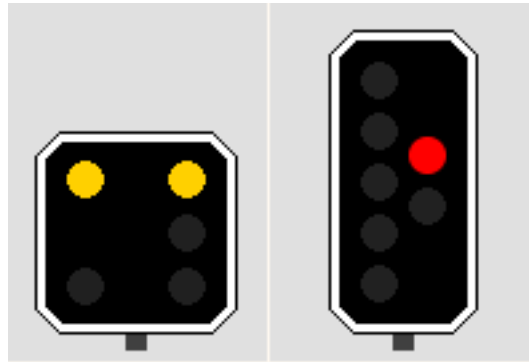


Figure 3.9: Left: Warning signal. Right: Main signal. Message: Stop

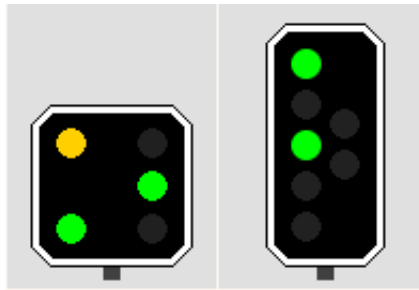


Figure 3.10: Message: Slow Approach(90 km/h)

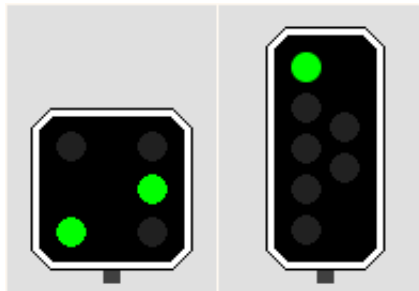


Figure 3.11: Message: Clear Line

Based on the signals we calculated the minimal braking time intervals (Algorithm 2):

Algorithm 2: Calculation of the minimal braking time interval

```

Input: entering node  $n_e$ , train route  $U$ 
Output: minimal braking interval start time  $T_s$  and end time  $T_e$ 
 $T_e$  = departure time at node  $n_e$  from ( $U$ );
if  $n_e$  = root node then
  |  $T_s$  = departure time at root node from ( $U$ );
end
else
  | node  $a$  = ancestor of  $n_e$ ;
  | while ( $a \neq$  root node) and ( $a$  does not contain a main signal) do
  | |  $a$  = ancestor of  $a$ ;
  | end
  | if  $a$  = root node then
  | |  $T_s$  = departure time at root node from ( $U$ );
  | end
  | else
  | | while ( $a \neq$  root node) and ( $a$  does not contain a warning signal) do
  | | |  $a$  = ancestor of  $a$ ;
  | | | end
  | | | if  $a$  = root node then
  | | | |  $T_s$  = departure time at root node from ( $U$ );
  | | | | end
  | | | else
  | | | |  $T_s$  = departure time at node  $a$  from ( $U$ );
  | | | | end
  | | | end
  | | end
  | end
end
return [ $T_s, T_e$ ];

```

The additional safety related time intervals compensate for uncertainties. Among them are:

- Time it takes to operate a track switch (7s)
- Reaction time of train drivers (5s)
- General Safety time buffer (12s)

Using these allocation time intervals at a resource conflicts can be detected easily: If two allocation time intervals overlap, the corresponding train routes are in conflict.

This conflict model supports the current safety system of the SBB [Gra07], as well as the prospective ETCS Level 2 safety model [Wik07]. After providing the information how the time intervals are calculated we can finally calculate the allocation time interval:

- The start time of the allocation interval T_s^a is equal to the start time of the minimal braking interval T_s^m : $T_s^a = T_s^m$
- The end time of the allocation interval T_e^a is equal to the end time of the occupation interval T_e^o with added safety times: $T_e^a = T_e^o + 7 + 5 + 12$

The next subsection describes how we can improve on the conflict modeling at the resource level.

3.3.2 Improving Conflict Modeling for a Resource

The conflict modeling in the tree conflict graph is simple: each time two train routes of distinct trains would allocate a resource at the same time, a conflict is introduced to the model. Drawing all the allocation time intervals of a resource during a period of time illustrates this concept of conflict modeling (see Figure 3.12).

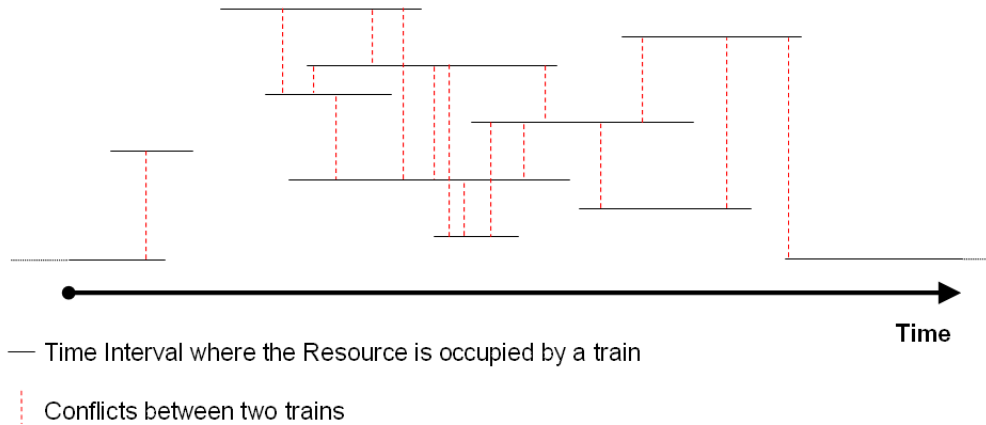


Figure 3.12: Allocation of a resource by several train routes

Instead of looking at pairs of overlapping allocation time intervals like in the previous approaches, a more sophisticated attempt would gather them into groups of conflicting intervals (see Figure 3.13).

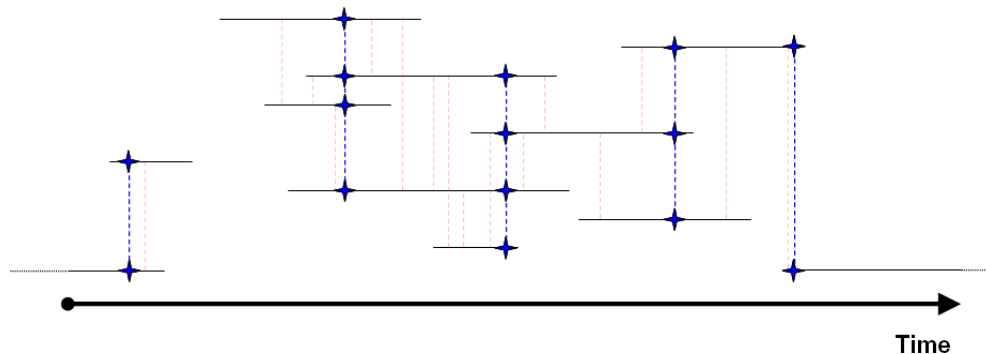


Figure 3.13: Improved Conflict Modeling

This idea has a huge impact on the conflict constraints of the resulting integer linear program:

- The constraints are much stronger: Each constraint corresponds now to a conflict group instead of corresponding to a conflict edge.
- The number of constraints is reduced significantly (Compare Table 5.2).

As the relaxation of the integer linear program will improve due to the stronger constraints, improving the conflict modeling is a key issue. To improve the conflict modeling, it is

necessary to find the minimum number of conflict groups covering all the conflicts between the allocation time intervals at this resource. This problem is equivalent to finding the minimum number of cliques to cover all the edges in the corresponding circular interval graph [Sch03]. For our modeling purposes we used a simple but efficient method:

1. Sort the start and end times of the allocation time intervals.
2. Walk through the sorted end times and build up the list of conflict cliques by looking at the already passed start times.

It is enough to consider the end times, since at that point the interval is closed and the invariant that all conflicts between the already covered intervals are detected is maintained. The same formulation in pseudo code:

Algorithm 3: Finding conflicting cliques of intervals from a set of intervals

Input: Set of allocation intervals $[T_s^a, T_e^a]$

Output: Set of conflict cliques C

Create a list L of pairs $(time \in \mathbb{N}, \text{boolean } is_endtime)$ from the allocation intervals

Sort T according to the first key (time);

foreach pair (t, e) in T **do**

if $e = \text{false}$ **then**

 | new_start_time := true;

end

else

if new_start_time := true **then**

 | Insert into C a conflict associated with times coming ahead of time t in T ;

end

 Remove (t, e) and the associated start time pair from T ;

 new_start_time := false;

end

end

return C ;

For better understanding the process is illustrated in the following example:

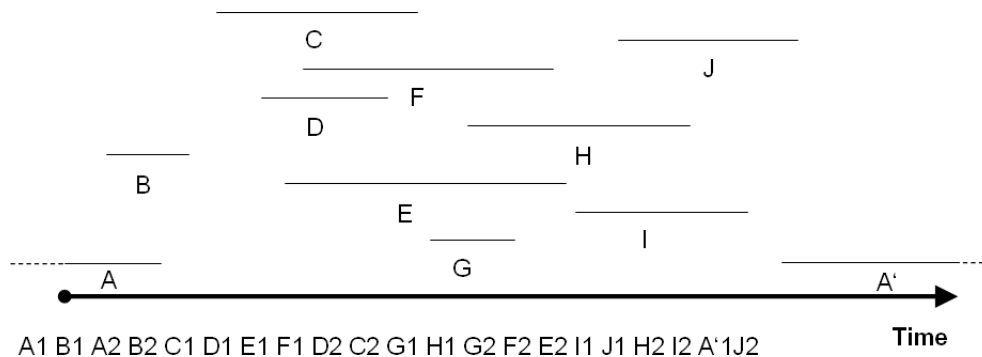


Figure 3.14: Algorithm 3: Sort start- and end-times

Continue with part 2 by walking through the end times:

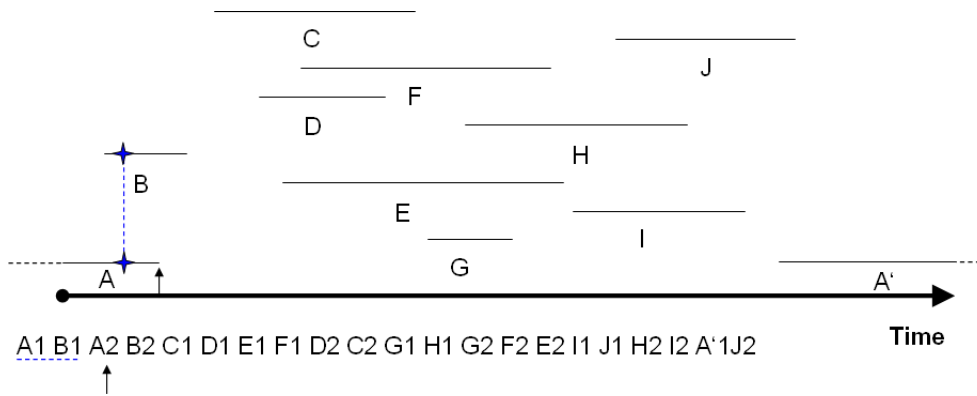


Figure 3.15: Algorithm 3: Look at first end time A_2 . The Start times A_1 and B_1 are gathered as a conflict clique.

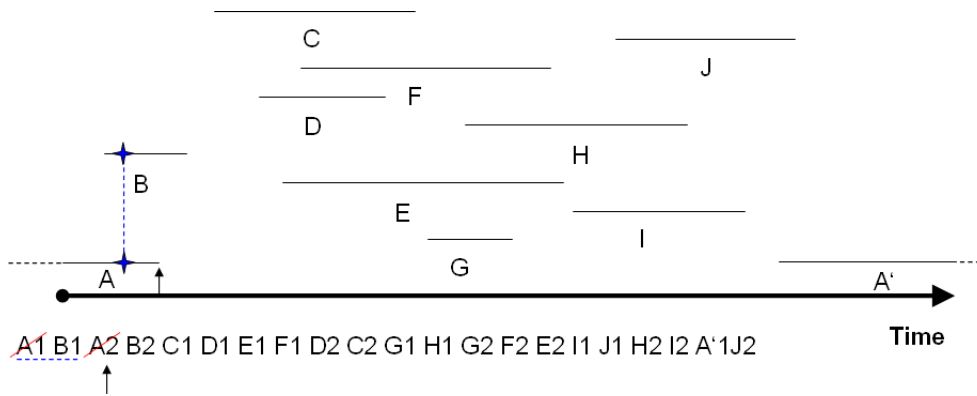


Figure 3.16: Algorithm 3: The interval A has ended, remove A from the list.

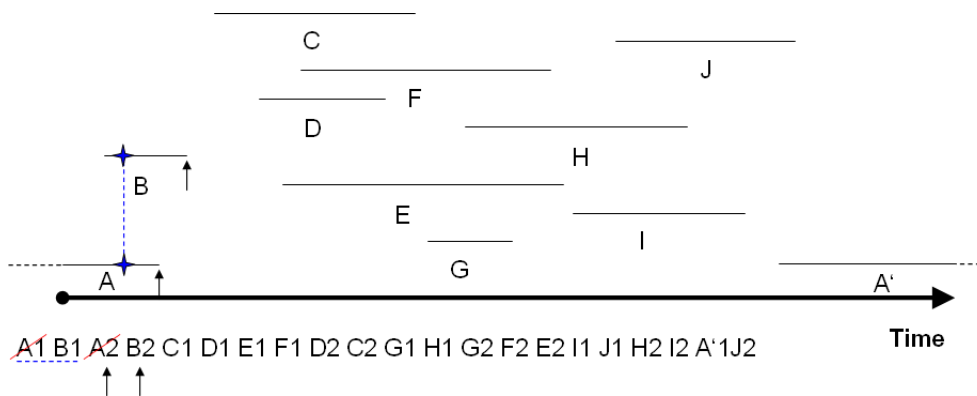


Figure 3.17: Algorithm 3: Look at the second end time B_2 . No new start time was inserted; Nothing to do.

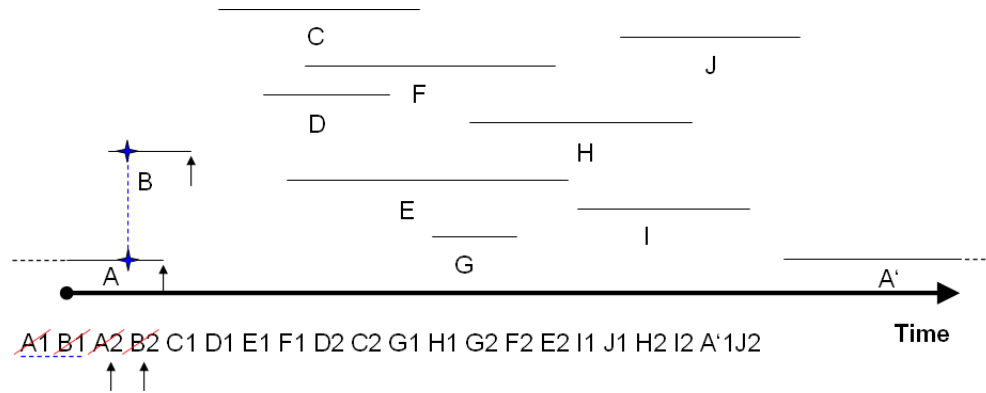


Figure 3.18: Algorithm 3: The interval B has ended, remove B from the list.

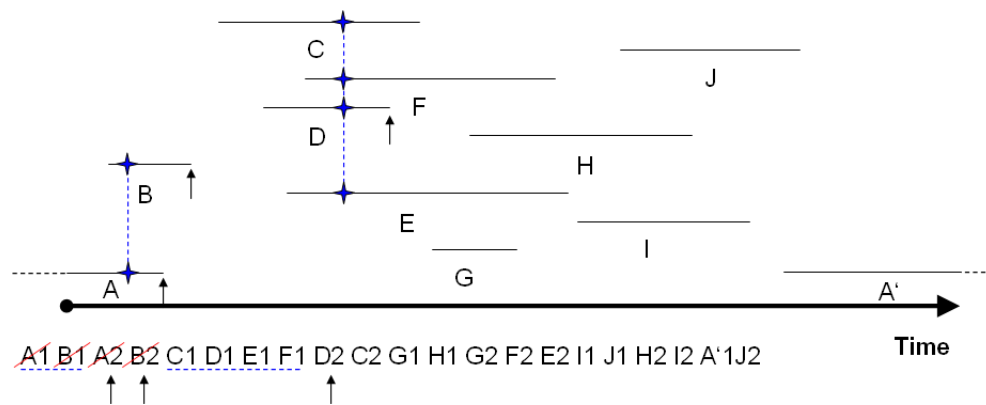


Figure 3.19: Algorithm 3: Look at third end time $D2$. The Start times $C1, D1, E1$ and $F1$ are gathered as a conflict clique.

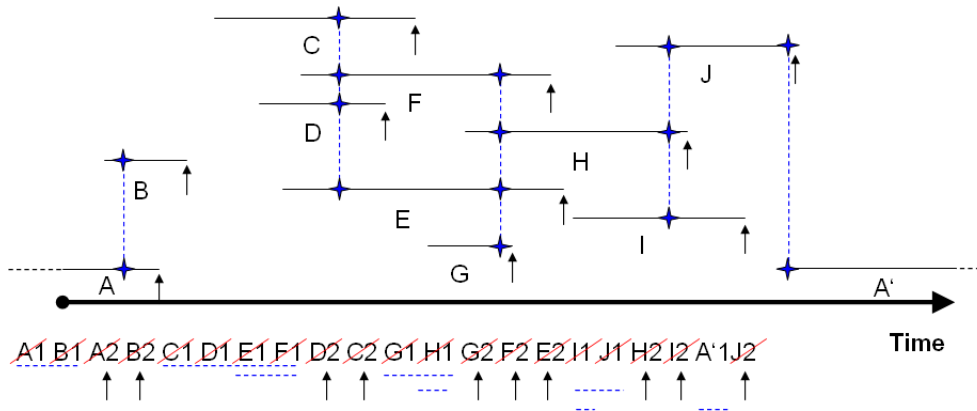


Figure 3.20: Algorithm 3: Final end time has been reached. All conflict cliques have been found.

The time complexity of algorithm 3 for the two parts:

- Sorting: With k intervals runs in logarithmic time $O(k \log k)$
- Grouping: Runs in linear time $O(k)$

Table 3.1: Algorithm 1: Runtime complexity

Note, that the described method fails to detect a conflict group in special cases:

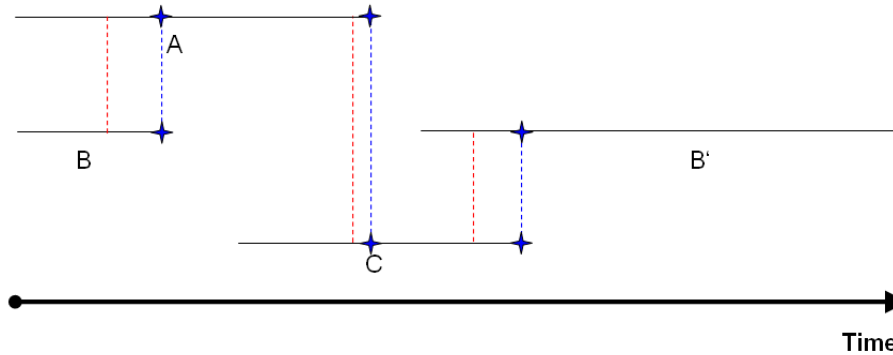


Figure 3.21: Special case, where the three clique consisting of the intervals A,B, and C is not detected

Since we are working with periodic allocation time intervals it can happen that a conflict group is not detected (see Figure 3.21). Typically, in practice the length of a period is one hour and the allocation time intervals are not longer than 3 minutes. Hence these special cases most likely will not occur. Nevertheless, it remains an open question of more theoretical interest if a polynomial time algorithm exists, which can find the minimum number of cliques which cover all the conflict edges in a circular interval graph.

3.3.3 Resource Tree Conflict Graph Model

The resource tree conflict graph consists of the same elements as a tree conflict graph with the exclusion of the conflict edges. For each resource we add a resource vertex to the model. After calculating the cliques for each resource with algorithm 3, these resource vertices are then connected to the nodes according to the calculated cliques. To distinguish the cliques of a resource, each clique receives its unique color in the illustrations. Note, that the resulting model is exactly equivalent to the tree conflict graph. In Figure 3.22 an example of a resource tree conflict graph is given. The integer linear program from the

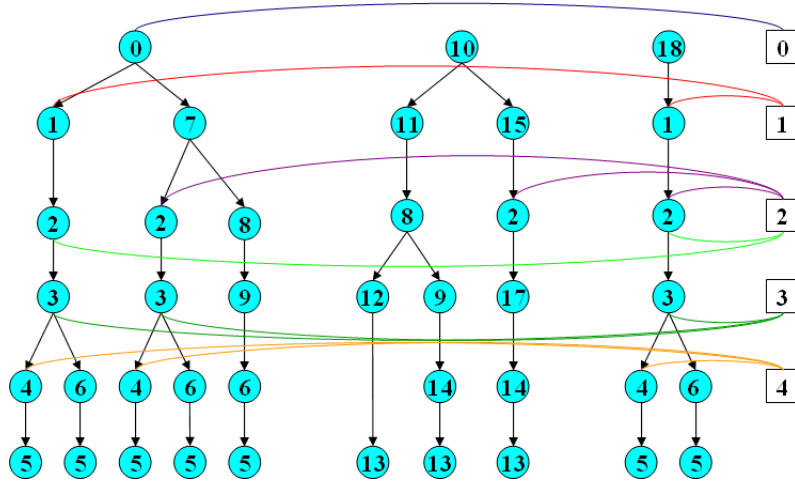


Figure 3.22: Example of a resource tree conflict graph. Each resource (on the right side) has its set of uniquely colored conflict cliques. Note that a vertex can be connected to the same resource several times, representing the fact that a time interval could be contained in more than one conflict clique.

previous tree conflict graph model is rewritten to include the improved conflict modeling:

Definition 3.10 (Resource Tree Conflict Graph - ILP Formulation)

Find x_{ij} such that:

1. $\sum_{x_{ij} \in S_i} x_{ij} = 1 \quad i = 1 \dots n$
2. $\sum_{x_{ij} \in T_i} x_{ij} = 1 \quad i = 1 \dots n$
3. $x_{ij} = \sum_{m: v_{ij} \rightsquigarrow v_{im}} x_{im} \quad \forall i, j : x_{ij} \notin \bigcup_{i=1}^n T_i$
4. $\sum_{(i,u) \in I} x_{iu} \leq 1 \quad \forall I \in F_k, \quad \forall \text{resources } k$
5. $x_{ij} \in \{0, 1\}$

where S_i is the set of edges connected to the source node of train i , T_i is the set of edges connected to the sink node of train i and F_k is the family of sets corresponding to conflict cliques of resource k . Note, that $v \rightsquigarrow w$ means that node v is ancestor of w and n is the number of trains.

Items (1.) - (3.) contain the flow preservation constraints: The inducted flow at each source is equal to one (1.) and the outflow at each sink is equal to 1 (2.). Between the sinks and the sources the flow is preserved (3.), where we do not have to consider the leafs. Item (4.) are the conflict clique constraints according to the gathered cliques of Algorithm 3 and (5.) restrict the variables to boolean values.

For improving the branch, bound and cut steps during the solving process it is advisable to introduce an objective function, instead of just solving a feasibility problem. Any objective function breaking the symmetry in the branch and bound tree is practical. For the computational results in this thesis the total train travel time of all trains was used as an objective function:

$$\sum_{i,j} x_{ij} \cdot t_{ij}, \quad \text{where } t_{ij} \text{ is the travel time from the ancestor node of } v_{ij} \text{ to } v_{ij}$$

Chapter 4

Model for Pulsed Train Scheduling with favored start times

In the pulsed train scheduling problem (Definition 2.7) each train has several (pulsed) selectable start times available. To receive a model for the pulsed train scheduling problem, we go back to the resource tree conflict graph:

1. For each selectable (pulsed) departure time of a train obtain a resource tree (see Figure 4.1).

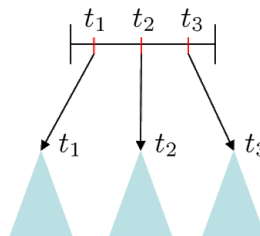


Figure 4.1: The three pulsed departure times are used to build the corresponding three resource trees.

2. Hook the created trees in the source node of the corresponding train (see Figure 4.2).

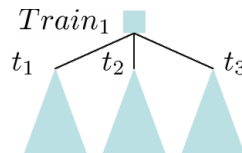


Figure 4.2: The created resource trees are hooked in the source node of the corresponding train.

For each source there are now several flow edges to resource trees with different start times. Since each source flow constraint associated with a train will only allow for maximal one

flow edge to be active, the resulting chosen departure time is the one with the active flow edge.

To support favored departure times, we change the objective function in the corresponding integer linear program:

$$\max \sum_{(i,j) \in S} w_{ij} \cdot x_{ij} \quad \text{where } S \text{ is the index set of the source flow variables.}$$

The weights are chosen according to the priorities of the departure times. We end up with the following integer linear program:

Definition 4.1 (RTCG with several weighted departure times - ILP Formulation)

$$\max \sum_{(i,j) \in S} w_{ij} \cdot x_{ij},$$

such that:

1. $\sum_{x_{ij} \in S_i} x_{ij} \leq 1 \quad i = 1 \dots n$
2. $\sum_{x_{ij} \in T_i} x_{ij} \leq 1 \quad i = 1 \dots n$
3. $x_{ij} = \sum_{m: v_{ij} \rightsquigarrow v_{im}} x_{im} \quad \forall i, j : x_{ij} \notin \bigcup_{i=1}^n T_i$
4. $\sum_{(i,u) \in I} x_{iu} \leq 1 \quad \forall I \in F_k, \quad \forall \text{resources } k$
5. $x_{ij} \in \{0, 1\}$

where S_i is the set of edges connected to the source node of train i , T_i is the set of edges connected to the sink node of train i and F_k is the family of sets corresponding to conflict cliques of resource k . Note, that $v \rightsquigarrow w$ means that node v is ancestor of w and n is the number of trains.

Items (1.) - (3.) contain the flow preservation constraints: The inducted flow at each source is less or equal to one (1.) and the outflow at each sink is less or equal to 1 (2.). Between the sinks and the sources the flow is preserved (3.), where we do not have to consider the leaves.

Item (4.) are the conflict clique constraints according to the gathered cliques of Algorithm 3 and (5.) restrict the variables to boolean values.

Since we allow now the flow to be less than 1, infeasibility can not occur. Due to the objective we schedule as many of the trains as possible, as well as we will prefer departure times with higher priority. Hence, if the solution of the integer linear program does not schedule all trains, it is impossible that all trains can be scheduled. This is much better than just a feasibility problem, since we still receive a solution even when we cant schedule all trains. The pulsed train scheduling problem with weighted departure times was not tested in practice for this thesis.

Chapter 5

Results for Train Routing Problem

We solved four test scenarios of the train routing problem, using the three presented models. For the test scenarios we used two variants of the track topology of the main station area Bern (Switzerland) with the associated timetable of 2003, as well as with an example of a prospective condensed timetable. The specifications of the system used for the calculations are:

- x86_64 architecture
- AMD Opteron 2.2 GHz
- 2 GB RAM

The programming language to implement the different models was C++.

5.1 Conflict Graph and Tree Conflict Graph

For the conflict graph model (**CG**) we used the randomized fixed point iteration proposed and implemented by Caimi et al. in [CHB05]. For the tree conflict graph model we solved the resulting integer linear program using the commercial solver ILOG CPLEX v9.1. The results are listed in the table 5.1.

Scenario	# conflicts CG	# conflicts TCG	CG time [s]		TCG time [s]	
			Construction	Solving	Construction	Solving
West 2003	70 000	15 500	1	5	4	< 1
West 2020	300 000	56 500	3	7	10	2
East 2003	740 000	85 500	7	15	50	4
East 2020	7 100 000	1 110 000	80	2400	930	180

Table 5.1: Results for the CG (using randomized FPI) and TCG (using ILP with CPLEX) Model

Based on these empirical results (table 5.1) we observed several aspects concerning the two models:

The tree conflict graph model has reduced the number of conflicts significantly compared to the conflict graph model. We expected this reduction in the number of conflicts due to

the binding of the conflicts to their location of occurrence. Nevertheless, the number of conflicts are still very high for big scenarios (more than a million).

We observe that the time needed to compute a solution has decreased. However, for big scenarios still around 15 minutes are necessary. This is explained by the necessary high number of conflict constraints for big scenarios.

You may have observed, that the construction time for the tree conflict graphs is higher compared to the construction time of the conflict graph. The reason for the increase in the construction time is the time necessary to search for conflicts. In both models each pair of nodes have to be checked for a possible conflict. However, due to the additional tree structure the number of nodes in the tree conflict graph is significantly higher when compared to the number of nodes in the conflict graph.

5.2 Resource Tree Conflict Graph

The same instances were used to solve the integer linear program of the resource tree conflict graph model. The corresponding integer linear program was solved again with the commercial solver ILOG CPLEX v9.1. The results are listed in table 5.2.

Scenario Bern	# nodes	# conflict cliques	Clique size			time [s]	
			Min	Average	Max	Construction	Solving
West 2003	12 200	750	2	12	132	< 1	1
West 2020	15 000	950	2	35	308	< 1	< 1
East 2003	46 600	3400	2	51	589	1	2
East 2020	57 800	4300	2	60	1128	2	2

Table 5.2: Results for the RTCG (using ILP with CPLEX) Model

Several aspects of these results (table 5.2) argue for the resource tree conflict graph model: due to the improved conflict modeling we expected a decrease in the number of conflicts. For the last instance the reduction was from more than one million conflict constraints down to approximately 4000 constraints, which is a lot more than we hoped for at the beginning.

The sizes of the conflict cliques are considerably large compared to the conflict pairs of the tree conflict graph model. These clique sizes are a measure for the strength of the conflict constraints, because the number of variables contained in a conflict constraint is equal to the clique size. Stronger constraints reduce the solution space of the LP relaxation without cutting out any integer solution and speed up the branch and bound method used by CPLEX.

Due to the reduced number of conflict constraints and their stronger formulation, solving the relaxed integer linear problem is for CPLEX much easier and less time consuming. For the biggest instance the solving time for the integer linear program was reduced from 15 minutes to two seconds.

Considering the construction time, we observe that the building of the resource tree conflict graph and the associated integer linear program is a lot faster. There are two reasons for this: The number of conflicts is considerably smaller and the conflict groups are easier to find (See Table 3.1 for the runtime complexity) since it is no more necessary to check each pair of resources.

Experience showed so far, that using this model one run first into memory problems instead of waiting for the CPU. We had only computational problems, when the memory capacity was reached.

Chapter 6

Results for Pulsed Train Scheduling Problem

For the pulsed train scheduling problem we used the biggest test scenario Bern East with the associated timetable of 2003. Using different numbers of departure time possibilities, we wanted to examine the scalability of the resource tree conflict graph model (See Table 6.1). For each train we allowed one starting time. By adding for each train more start time possibilities, more and more trees are created in the model. At the end 1.4 GB Memory was needed to store the trees and the integer linear program.

#Start Times	#nodes	#conflict cliques	Clique Size			time[s]	
			Min	Average	Max	Construction	Solving
1	175 000	3 900	2	44	851	3	3
2	350 000	7 900	2	46	851	4	5
4	700 000	14 500	2	54	851	6	10
6	1 050 000	22 300	2	68	1068	9	18
8	1 400 000	27 200	2	74	1068	11	27
10	1 750 000	34 800	2	80	1068	14	35

Table 6.1: Results for the RTCG (using ILP with CPLEX) Model

We compare the results of the pulsed train scheduling problem (table 6.1) with the results of the train routing problem (table 5.2):

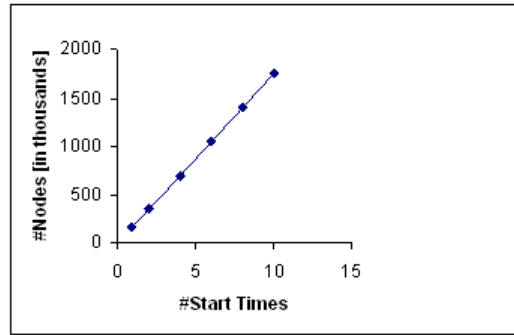


Figure 6.1: Pulsed Train Scheduling (Bern East 2003): Number of nodes

The number of nodes increases linearly proportionally to the number of start times, since for each start time a resource tree has to be built (Fig. 6.1). This is arguably important, since the memory usage does not increase dramatically. Besides the nodes of the tree, the conflict cliques are also a factor for memory usage:

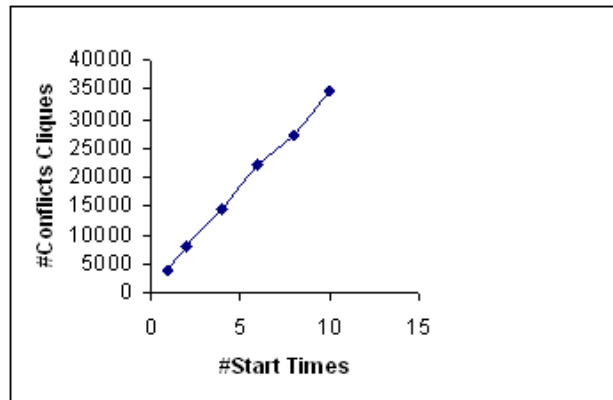


Figure 6.2: Pulsed Train Scheduling (Bern East 2003): Number of conflict cliques

The number of conflict cliques grows in this example linearly as well (Fig. 6.2). Although the number of conflicts grow linearly in this example, it could grow faster. Imagine if one would add a new start time for a tree and the resulting train routes are in conflict with other routes at almost every node and hence the number of conflicts would increase suddenly. Nevertheless, compared to the number of nodes and flow constraints the memory consumption of conflict constraints is negligible.

The average conflict clique size is steadily increasing, since more allocation time intervals cause more conflicts (Fig. 6.3). This results in stronger constraints in the integer linear program and hence in an overall improvement of the solution method.

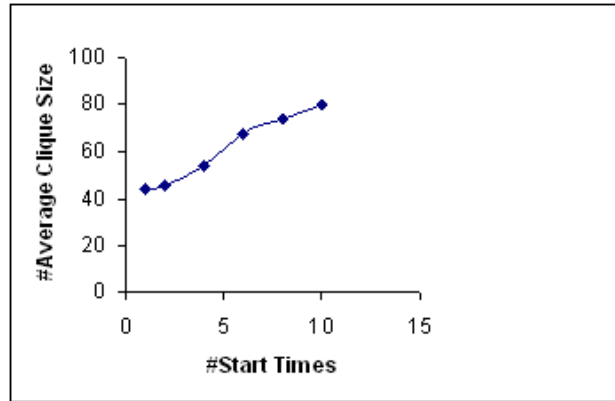


Figure 6.3: Pulsed Train Scheduling (Bern East 2003): Average clique size

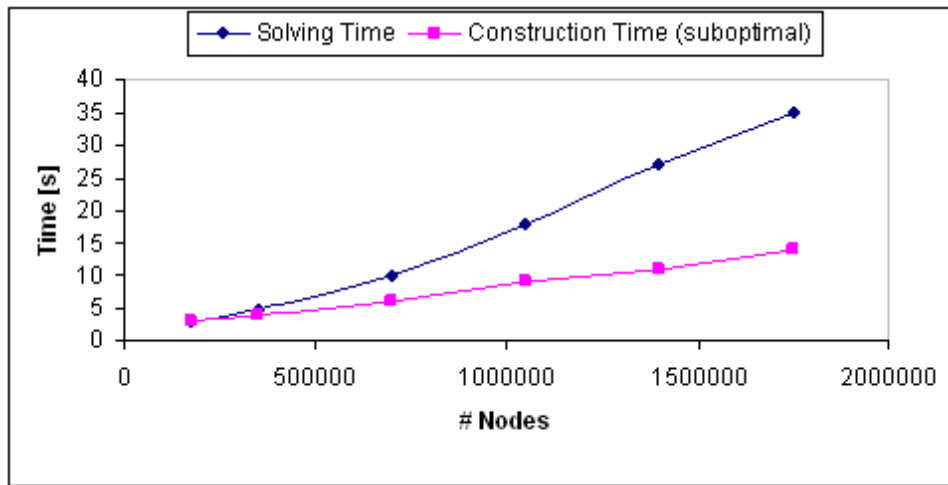


Figure 6.4: Pulsed Train Scheduling (Bern East 2003): Computation times

The various observations above explain that the computation time seems to grow linear with increasing problem size for our instances (Fig. 6.4). Hence, the resource tree conflict graph model seems to be scalable to several departure times. However, we do not have any guarantee for the scalability.

Chapter 7

Conclusion and Outlook

We have considered the train routing problem and the pulsed train scheduling problem in a main station area. Based on improvements in the structures of existing models, we were able to solve these two problems for four scenarios based on data of the main station area Bern, Switzerland. For the used data the pulsed train scheduling problem is now exactly solvable in a very small amount of time. Hence, railway companies should consider using this model in the future for scheduling pulsed trains in the main station areas.

We also think that this model could be a basis for future research:

- It is unclear how the model will perform for regions with less rail tracks and track switches. A possible test case scenario could be a long track line with few routing possibilities and several trains with different speed profiles.
- Is the model scalable to larger areas?
- How could we include stability measures in train schedules as a objective for the solution approach?
- How could the model be connected to train scheduling models which work on the nationwide level (e.g. Periodic Event Scheduling Problem)?
- Could the ideas of the model be used to come up with a model for other traffic related problems, e.g. air traffic?

Bibliography

- [AMO93] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin, *Network Flows*, Prentice Hall, 1993.
- [CC03] M. Carey and S. Carville, *Scheduling and platforming trains at busy complex stations*, *Transportation Research* **1** (2003), 195–224.
- [CHB05] Gabrio Caimi, Thomas Herrmann, and Dan Burkolter, *Finding Delay-tolerant Train Routing through Stations*, *Operations Research Proceedings 2004* (H. Fleuren, D. den Hertog, and P. Kort, eds.), GOR, Springer, 2005, pp. 136 – 143.
- [EVS05] Matthias Ehrgott, Rafael Velasquez, and Anita Schöbel, *A Set-packing Approach to Routing Trains Through Railway Station*, Preprint nr. 2005-36, Georg August Universität Göttingen, 2005.
- [Gra07] Thomas Graffagnino, *Méthode de Calcul des Temps de Blocage*, v. 10.0, Swiss Federal Railways, 2007.
- [HC06] Thomas Herrmann and Gabrio Caimi, *Model and algorithm for rerouting delayed trains online*, Euro XXI Conference, 2006.
- [Her05] Thomas Herrmann, *Train Routings through Station Areas and stability of Timetables*, Ph.D. thesis, ETH Zurich, 2005.
- [HKL05] D. Huisman, L. Kroon, R. Lentink, and M. Vromans, *Operations Research in Passenger Railway Transportation*, *Statistica Neerlandica* **59** (2005), no. 4, 467–497.
- [Mon92] Markus Montigel, *Representation of track topologies with double vertex graphs*, *Computers in Railway* (Washington D.C.) (T.K.S. Murthy, F.E. Young, S. Lehmann, and W.R. Smith, eds.), Computational Mechanics Publications, vol. 2, 1992.
- [Roo06] Samuel Roos, *Bewertung von Knotenmanagement-Methoden für Eisenbahnen*, Master's thesis, ETH Zurich, 2006.
- [SBB05] SBB, <http://www.sbb.ch>, 2005.
- [Sch03] Alexander Schrijver, *Combinatorial optimization*, Springer, 2003.
- [Wik07] Wikipedia, <http://en.wikipedia.org/wiki/ETCS>, 2007.
- [ZKR⁺96] Peter J. Zwaneveld, Leo G. Kroon, H. Edwin Romeijn, Marc Salomon, Stéphane Dauzère-Pérès, Stan P.M. Van Hoesel, and Harrie W. Ambergen, *Routing Trains through Railway Stations: Model Formulation and Algorithms*, *Transportation Science* **30** (1996), no. 3, 181–194.