Resolving infeasibilities in railway timetabling instances

Gert-Jaap Polinder[1], Leo Kroon[1,2], Karen Aardal[3], Marie Schmidt[1] and Marco Molinaro[4]

[1]*Rotterdam School of Management, Erasmus University Rotterdam*
[2]*Process Quality and Innovation, Netherlands Railways, Utrecht, The Netherlands*
[3]*Delft Institute of Applied Mathematics, Delft University of Technology, The Netherlands*
[4]*Computer Science Department, PUC-Rio, Brasil*

Abstract

One of the key assumptions of timetabling algorithms is that a solution exists that meets the pre-specified constraints, like driving times, transfer constraints and headway constraints. If this assumption is satisfied, in most cases a timetable can be found rapidly. Nowadays, railways are being used more intensively, which leads to a higher utilization of the network. Due to this increased utilisation, capacity conflicts occur, so that no feasible solution to the timetabling models can be found, without making subtle but non-trivial changes to the initial input. Resolving these conflicts is essential for railway companies with high utilization of infrastructure. In this paper, we consider infeasible timetabling instances together with a list of allowed modifications of the constraints. We iteratively identify local conflicts in these instances and resolve them by adapting some of the constraints, until there are no more conflicts. The adaptations of the constraints are changes in the right-hand sides that we try to make as small as possible but that resolve the infeasibility. We empirically show that our method can be improved by enriching the initial minimal conflicts found with more constraints. In order to keep the problems tractable, an iterative procedure is used to find solutions to subproblems corresponding to conflicts in the complete timetabling instance. In a case study on instances from the Dutch railway network, we show that these instances can be made feasible within a few minutes.

† In memoriam of Leo G. Kroon who passed away on September 14th, 2016.

1 Introduction

In Europe, many public transportation railway companies operate a cyclic timetable. This means that the timetable is repeated every time period, usually every hour. For example, in the Netherlands during daytime, every hour at .05, an intercity leaves Rotterdam heading for Utrecht [NS, 2017]. A timetable is also generally referred to as a *schedule*, which we use throughout this paper.

A mathematical model that is commonly used for finding such a cyclic (also called periodic) schedule is the Periodic Event Scheduling Problem (PESP) as introduced in Serafini and Ukovich [1989]. This problem can be stated as follows:

Definition 1.1 (PESP). *Given an instance $\mathcal{I}$, consisting of a set V of events, a set $A \subseteq V \times V$ of constraints, intervals $[l_{ij}, u_{ij}]$ for all $(i,j) \in A$ and a period length T, the Periodic Event Scheduling Problem is to find a feasible periodic schedule, that is, event times $\pi : V \to [0,T)$ satisfying*

$$\pi_j - \pi_i \text{ modulo } T \in [l_{ij}, u_{ij}] \qquad \forall\, (i,j) \in A. \tag{1}$$

If such a periodic schedule exists, we call the instance $\mathcal{I}$ feasible. Else, we call it infeasible. Note that any PESP instance can be scaled in such a way that $0 \leq l_{ij} < T$ (cf. Peeters [2003]). Next to this, we may assume that $u_{ij} - l_{ij} < T$, otherwise the constraint would be redundant.

In the railway timetabling context, the events V correspond to departures or arrivals at some location in the rail network. The constraints A state relations in time between pairs of events, that is, the difference in time between two events i and j should be within a T-periodic interval $[l_{ij}, u_{ij}]$. The constraints model bounds on trip times, stopping times, safety distances between pairs of trains and desired connection times. To show what a PESP instance can look like in a railway context, consider the following example.

Example 1.1. The situation in this example is that trains 1 and 2 somewhere in their journey drive from station s to station s', while sharing the same track. To determine a schedule for this situation, we need to plan the departure times from station s and s' and the arrival times at station s' of these trains. This leads to determining values for $\pi^s_{1,d}$, $\pi^s_{2,d}$, $\pi^{s'}_{1,a}$, $\pi^{s'}_{2,a}$, $\pi^{s'}_{1,d}$ and $\pi^{s'}_{2,d}$. Here, π denotes the event time, the superscript shows the station of the event. The subscript shows the train number and the event type: a departure (d) or arrival (a).

The trip time of train 1 is at least 6 and at most 7 minutes, so it is in the interval $[6, 7]$. The trip time of train 2 resides in the interval $[7, 8]$. Since trip times are allowed to vary in the specified intervals, we refer to this as *variable trip times*. The trains stop for exactly one minute at station s'.

Next, we have an 'ABS' event, which is a dummy event that does not correspond to a specific train. We use this event to ensure that certain events will happen within a given time interval, if this is desired. For example, if train 1 has to leave station s in the interval $[18, 22]$, we add the constraint $\pi^s_{1,d} - \pi_{ABS}$ modulo $T \in [18, 22]$. Note that once a feasible schedule is obtained for $\mathcal{I}$, every event time can be shifted by $\delta < T$ minutes, resulting in a new timetable, with the same time differences on the constraints, but with different event times. By doing so, we can find δ such that $\pi_{ABS} = 0$ and hence the desired times for the events that have to happen at a specific time are ensured.

Next to the given events, we consider the following constraints:

$$\pi^{s'}_{1,a} - \pi^s_{1,d} \in [6, 7]_{60} \tag{2a}$$
$$\pi^{s'}_{2,a} - \pi^s_{2,d} \in [7, 8]_{60} \tag{2b}$$
$$\pi^{s'}_{1,d} - \pi^{s'}_{1,a} \in [1, 1]_{60} \tag{2c}$$
$$\pi^{s'}_{2,d} - \pi^{s'}_{2,a} \in [1, 1]_{60} \tag{2d}$$
$$\pi^s_{2,d} - \pi^s_{1,d} \in [30, 30]_{60} \tag{2e}$$
$$\pi^s_{1,d} - \pi_{ABS} \in [18, 22]_{60} \tag{2f}$$
$$\pi^s_{2,d} - \pi_{ABS} \in [48, 52]_{60} \tag{2g}$$
$$\pi^s_{2,d} - \pi^s_{1,d} \in [h, 60 - h]_{60} \tag{2h}$$
$$\pi^{s'}_{2,a} - \pi^{s'}_{1,a} \in [h, 60 - h]_{60} \tag{2i}$$
$$\pi^{s'}_{2,d} - \pi^{s'}_{1,d} \in [h, 60 - h]_{60}. \tag{2j}$$

Note that in this example we use $T = 60$.

In this example, Constraints (2a) and (2b) correspond to trip time constraints, forcing the trip time between the departures at two stations to be within some interval. Although both trains use the same track, it is possible that their trip times are different, for example when different types of trains are used. Constraints (2c) and (2d) correspond to dwell time constraints, forcing the dwell time in station s' to equal one minute.

Next, these two trains have to drive in a 'nice' pattern, i.e., they have to depart at the first stations 30 minutes apart from each other, giving constraint (2e). Next to this, suppose regulations require train 1 (train 2 resp.) to leave station s in the interval $[18, 22]$ ($[48, 52]$ resp.), which is modelled by (2f) and (2g). In order to guarantee safety, a minimum time span of h minutes is to be respected between the departure and arrival of the trains at the station (generally $h = 3$). This leads to constraints (2h) and (2j), to guarantee safety upon departure, and (2i) to guarantee safety upon arrival.

Note that a schedule for this example can be found easily, for example $\pi^s_{1,d} = 20$, $\pi^{s'}_{1,a} = 26$, $\pi^s_{2,d} = 50$ and $\pi^{s'}_{2,a} = 57$, where we used $h = 3$.

The example stated above uses both departure and arrival times. When travel times are assumed to be fixed, it is possible to merge pairs of trip time and dwell time constraints and state PESP instances solely in departure events: When trip times are fixed, the width of the intervals in (2a) and (2b) is zero. Then, knowing the time of a departure event, the time for the corresponding arrival event follows immediately. Planning only departure events has the advantage that fewer event times have to be determined, thus reducing the complexity of the problem. However, assuming fixed travel times implicitly leaves less flexibility in the constraint set, since there is no freedom in choosing a travel time. Liebchen and Möhring [2007] describe under which circumstances this flexibility loss can be overcome. Note that when modelling single tracks, fixed travel times are needed to model safety constraints correctly. Furthermore, fixed trip time reduce the solution space and can thus decrease running time of solution methods.

Transforming Example 1.1 to a situation with fixed travel times, can for example lead to the following:

Example 1.2. We now assume that train 1 uses 6 minutes to travel from s to s', while train 2 needs 7 minutes. When stating the constraints solely in departure events, we can omit the departure/arrival notation. Furthermore, we use $h = 3$ and thus arrive at the following set of constraints:

$$\pi_1^{s'} - \pi_1^s \in [7,7]_{60} \tag{3a}$$
$$\pi_2^{s'} - \pi_2^s \in [8,8]_{60} \tag{3b}$$
$$\pi_2^s - \pi_1^s \in [30,30]_{60} \tag{3c}$$
$$\pi_2^{s'} - \pi_1^{s'} \in [30,30]_{60} \tag{3d}$$
$$\pi_1^s - \pi_{ABS} \in [18,22]_{60} \tag{3e}$$
$$\pi_2^s - \pi_{ABS} \in [48,52]_{60} \tag{3f}$$
$$\pi_2^s - \pi_1^s \in [3,57]_{60} \tag{3g}$$
$$\pi_2^s - \pi_1^s \in [2,56]_{60} \tag{3h}$$
$$\pi_2^{s'} - \pi_1^{s'} \in [3,57]_{60}. \tag{3i}$$

Constraints (3a) and (3b) arose by merging a trip time constraint with a dwell time constraint. The constraints that include a trip time part and a positive dwell time, are referred to as 'trip-dwell constraints'. If the dwell time is zero, i.e., the train does not stop at this station, these constraints are referred to as trip constraints. Constraints (3c) and (3d) impose the synchronisation between two departure at the same station. Constraints (3e) and (3f) require the trains to depart within a given interval. Finally, we have constraints guaranteeing safety upon departure ((3g) and (3i)) and upon arrival (3h). The reason that the interval in (2i) is not $[3, 57]$ is because of the trip time difference between the two trains.

Note that, because of the size of this instance, there is no freedom in finding a schedule, everything is already determined. In real-life instances, which are too large to be considered here in an example, this is not the case.

We can represent PESP instances as a graph $G = (V, A)$, where nodes model events V and arcs model constraints A. A PESP-graph representation for the instance in Example 1.2 is shown in Figure 1.

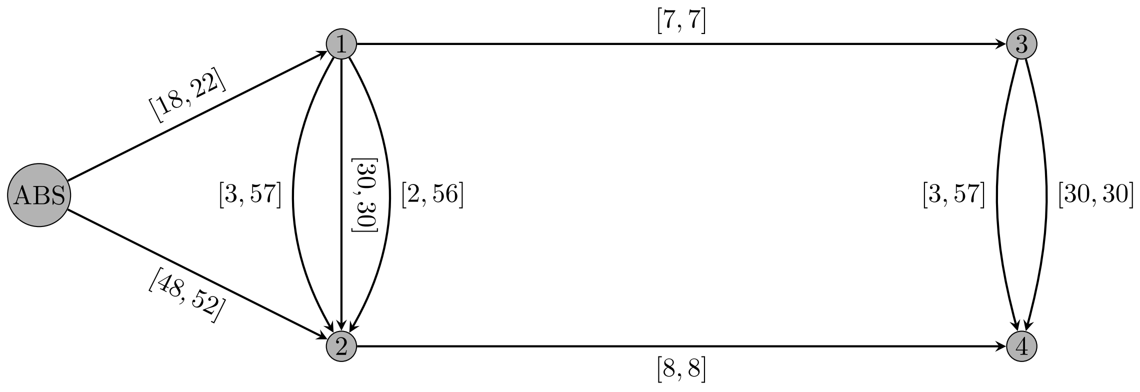


Figure 1: PESP graph for Example 1.2

There is a one-to-one correspondence between a node/arc and a PESP event/constraint. Therefore, when it fits better into the context, we sometimes refer to 'nodes' (resp. 'arcs') when talking about PESP events (resp. constraints).

In Example 1.2 and the corresponding constraint graph in Figure 1, we see that there is no solution π to the given set of constraints, because constraints (3a) - (3d) can never be satisfied simultaneously. We say they form a *conflict*.

1.1 Problem statement

In this paper, we deal with PESP instances from railway timetabling for which no feasible schedule exists. That means, the instances contain sets of constraints that together form a conflict, these sets together prohibit the existence of a feasible solution. The focus of this paper is not to *solve* PESP but to *resolve the infeasibilities* in infeasible PESP instances. This 'resolving' is done by changing bounds of the PESP-constraints as little as possible where this is needed.

Formally, we define the problem that we resolve in this paper as follows.

Definition 1.2 (Infeas-PESP). *Let $\mathcal{I}$ be an infeasible PESP instance, with constraint graph $G = (V, A)$. For each $a \in A$, let l_a (u_a) denote lower (upper) bounds of the constraints and τ_a^l (τ_a^u) the given maximum allowed deviations of these bounds. The task is to find new constraint bounds $l'_a = l_a - s_a^l$ ($u'_a = u_a + s_a^u$), where $s_a^l \in [0, \tau_a^l]$ ($s_a^u \in [0, \tau_a^u]$), such that, with respect to these new bounds, a feasible schedule $\pi : V \to [0, T)$ exists, or to decide that no such change exists.*

Note that it might not always be possible to find changes to the constraints satisfying the given bounds τ. In that case, no resolution of a conflict is possible.

If a resolution of the conflicts exists, we aim at finding one that violates the original constraints as little as possible. Since PESP constraints model different real-life constraints, like driving and dwell times, transfer times, or safety distances, violation of one constraint may be more undesirable than of another. For this reason, we introduce weights on the constraint violations, which we try to minimize heuristically.

The remainder of this paper is structured as follows. In Section 2 we review the literature on PESP and conflict solving and state our contribution. In Section 3, we describe the above outlined procedure in more detail. First, the notion of a *conflict* is formally defined. Next, several models are formulated to resolve a conflict in Section 3.2. Methods to add more relevant constraints to conflicts are shown in Section 3.2.4. The iterative algorithm to subsequently resolve conflicts is proposed in Section 3.3. Computational results are presented in Section 4. We end with a conclusion and discussion in Section 5.

2 Literature review and contribution

2.1 Periodic event scheduling

As was mentioned in the introduction, many of the approaches for finding a feasible cyclic railway schedule are based on the Periodic Event Scheduling Problem [Serafini and Ukovich, 1989], see Definition 1.1. That this model is very well suitable to model requirements for periodic timetabling is shown for example in Peeters [2003], Liebchen and Möhring [2007]. In Peeters [2003], an extensive overview is given on how to model practical requirements to a periodic schedule using PESP. It further contains an overview of relevant literature concerning PESP.

Caimi et al. [2017] provide a clear overview of the different models that can be used for (mainly periodic) scheduling, and how these are applied in practice. They claim that PESP is the most widely used model, many of the other models for periodic timetabling are variants of this model. The PESP framework can be used to model a lot of requirements, but not everything can be modelled in purely PESP constraints, as is shown in Liebchen and Möhring [2007], where it is shown which kind of requirements can be incorporated in the PESP framework and which cannot.

Since the introduction of PESP, much research has been devoted to the model and how to find solutions to it.

PESP constraints can easily be translated to mixed integer programming (MIP) constraints by introducing an integer variable p_{ij} for each $(i, j) \in A$ to reformulate the modulo operator in (1). We obtain the PESP-IP as (cf. Peeters [2003]):

Definition 2.1 (PESP-IP). *Given a set V of events, a set $A \subseteq V \times V$ of PESP-constraints, find a feasible solution to*

$$l_{ij} \leq \pi_j - \pi_i + T p_{ij} \leq u_{ij}, \qquad \forall\, (i, j) \in A \tag{4a}$$

$$\pi_i \in [0, T - 1] \qquad \forall\, i \in V \tag{4b}$$

$$p \in \mathbb{Z}^m_{\geq 0}. \tag{4c}$$

Then, these MIP models can be solved using state-of-the-art solvers like IBM ILOG CPLEX or Gurobi. This approach allows to find feasible schedules that are optimal according to some objective function. However, for real life instances, this approach often does not find solutions within reasonable time. Several other solution methods exist which are often able to solve real-life instances within reasonable time (cf. Schrijver and Steenbeek [1993], Großmann et al. [2012], Odijk et al. [2002]). These solution methods are not aimed at finding the best schedule, but at finding a feasible schedule or detecting that the underlying PESP instance is infeasible.

In Odijk [1996], a cutting plane approach to solve PESP is proposed and applied on a small part of the Dutch railway network. It starts with (PESP-IP) and a set of vectors for all possible values for the p-variables in (4a). Based on cycles in the constraint graph, restrictions can be added on the possible values that the p-values can take. Eventually, the algorithm ends up with a feasible p-vector (and hence a schedule) or proves that the model is infeasible.

Nachtigall [1999] further formalized this concept and introduced a reformulation of PESP as a MIP, known as the Cycle Periodicity Formulation (CPF). We restate this formulation in the Appendix C (see Definition C.1). An advantage of (CPF) over (PESP-IP) is that it uses fewer integer variables and equality constraints instead of inequality constraints which has advantages in a branch-and-bound procedure when solving the models. For a further discussion and comparison of different PESP formulations, see for example Liebchen et al. [2008].

2.2 PESP conflicts

The problem of running into infeasibilities when solving PESP has been mentioned before. Kroon and Peeters [2003] noted that this problem can be caused by assuming fixed trip times which is an assumption that is often made in railway timetabling (cf. Kroon and Peeters [2003], Kümmling et al. [2015a]). In order to incorporate more flexibility into the mathematical models that are used, the authors state under which necessary and sufficient conditions trip times can be allowed to vary. This provides a bit more flexibility in the planning processes, but in practice conflicts still arise in busy railway networks. Recently, also Kümmling et al. [2015a] considered the problem of infeasible PESP instances. More technical details on their approach are given inGroßmann et al. [2015]. The authors try to make the PESP instance feasible by applying a binary search heuristic and a MaxSAT approach. This method finds changes to upper bounds of constraints, and in some cases lower bounds as well, in order to determine if a feasible schedule exists. The authors use a function that assigns a weight to each constraint, indicating how expensive it is to change this constraint. The weight is set to infinity if a constraint cannot be changed. In their paper, no specific weight function is given. The authors, however, claim that in railway instances typically only dwell time constraints and connection constraints can be relaxed. This is due to the fact that PESP does not allow for dependencies between constraints (see also Kümmling et al. [2015b]), which is the case when trip times are not assumed to be fixed. They test their model on several instances, among which the largest one contains all the high speed lines and some of the most important regional train lines in Germany.

2.3 Our contribution

In this paper, we propose a methodology to resolve infeasible PESP instances by altering PESP constraints.

Our method is applicable both to PESP instances with variable trip times, and to PESP instances with fixed trip times. In the former case, the altering of the bounds corresponds to a widening of the interval which bounds the time difference between two events. For PESP with fixed trip times, we relax all but the fixed trip time constraints in the above-described way. For the trip time constraints, we do not consider relaxation of the corresponding interval, but instead a change of the trip time. To take implicit interdependencies in the constraint set into account, we introduce relations between the PESP constraints.

We propose a mixed-integer program (MIP) that is able to alter PESP constraints in both ways described above, and can hence solve conflicts both in PESP instances with variable and with fixed trip times. While in theory, our MIP model could be used to resolve all conflicts in the PESP instance at the same time, infeasible PESP instances encountered in (timetabling) practice are too large to be resolved in one go.

To be able to resolve large PESP instances, we propose a methodology that iteratively detects minimal conflicts (that is, a minimal set of constraints that cannot be satisfied simultaneously), enriches them in a heuristic way, and solve them using the MIP. This procedure is repeated until all conflicts are resolved.

In our computational experiments, we systematically analyse and compare which methods of enriching a conflict work well. We test our approach on timetabling instances from Netherlands Railways (NS), which is one of the most intensively used railway networks in the world, with over 75,000 constraints in the PESP model. Our method is able to resolve conflicts, even in such a large and highly utilized network, within minutes.

We ran our computational experiments based on weights defined after communication with planners from NS. However, the short computation times of our method would even allow to use our method in an interactive way: Planners could set weights based on a first judgement of the importance of constraints, and then adjust them based on the result found, to guide the solution into the desired direction.

3 Conflict resolving

PESP instances that arise in practice can be infeasible. This means they contain a set of constraints that cannot be satisfied simultaneously. The aim is to make such instances feasible and in this section we describe our approach to achieve this goal. (See Section 1.1 for a formal definition of the problem).

We propose an iterative procedure, as follows:

- Identify a local conflict (an exact definition is given in Section 3.1).
- Resolve the conflict using a MIP that minimizes a weighted sum of the changes in the network constraints (Section 3.2).
- Carefully select additional constraints and events to be added to this local conflict, in order to find a better resolution of this conflict (discussed in Section 3.2.4)
- Modify the constraints based on the solution provided by the MIP and search for a new conflict (go back to the first step)
- As soon as we do not encounter a conflict anymore, we try to further optimize the changes that are made (described in Section 3.3.4).

This is detailed in the following sections.

3.1 Conflicts in PESP

In order to be unambiguous in the terms we are using, we give a definition of what we mean by the term 'conflict':

Definition 3.1 (Conflict)**.** *A* conflict *is a set of events and PESP constraints such that no schedule exists satisfying all the constraints.*

In line with the definition of a conflict, we have the following definition of a minimal conflict:

Definition 3.2 (Minimal conflict)**.** *A* minimal conflict *is a conflict that has the additional property that no single constraint can be removed without creating a feasible set of constraints.*

Note that minimal conflict does not mean it is of minimum size. In fact, minimal conflicts can arbitrarily differ in size, as the next example shows.

Example 3.1. In Figure 2 a conflict graph is shown with events $V = \{0, 1, \dots, N\}$. For each $i = 1, 2, \dots, N$ we have the constraint $\pi_i - \pi_{i-1} \in [1/N, 1/N]_T$. Furthermore, we have the constraints $\pi_N - \pi_0 \in [1, 1]_T$ and $\pi_N - \pi_0 \in [2, 2]_T$.

Clearly, the latter two constraints form a conflict, which is denoted by C_1. However, the constraints $\pi_N - \pi_0 \in [2, 2]$ together with the set of constraints $\pi_i - \pi_{i-1} \in [1/N, 1/N]$ for all $i = 1, \dots, N$ form a conflict, too. Denote this conflict by C_2. Both conflicts are minimal: removing any constraint from the conflicts would resolve these conflicts. However, the number of constraints in each conflict is different.

So by definition, any infeasible PESP instance contains at least one conflict. There exist different methods on how to identify conflicts, see, e.g., Kümmling et al. [2015b]. In this paper, we describe a methodology to make the PESP instance feasible by resolving the conflicts through a change in constraint bounds, as defined in Definition 1.2.

When modelling railway scheduling instances as PESP problems, PESP constraints describe different types of real-life railway timetabling constraints, like trip time constraints, dwell times

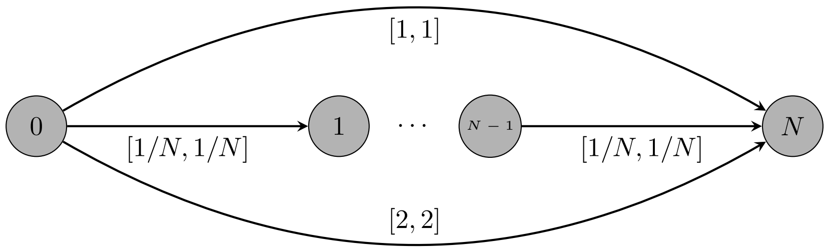


Figure 2: Minimal conflicts

constraints (or trip-dwell times constraints in the condensed model with fixed trip times), safety constraints, or so-called market requirements. Market requirements refer to *transfer constraints*, which state the arrivals and departures of different trains at the station need to be matched to allow for convenient transfers, and *synchronization constraints*, which lead to an equal spacing in time of trains of the same line.

By how much one is allowed to deviate from these constraints and how much a deviation is penalized depends on the instance, and should in practice be decided based on discussion with the train operator.

3.2 Resolving a single conflict

In order to find a model to resolve conflicts in PESP, note that each PESP constraint states that the difference in time between pairs of events should be within a given periodic interval. In case of a conflict, these intervals are too small, thus preventing the existence of a solution to PESP. Therefore, the bounds have to be changed in order to obtain a feasible solution. The goal of this section is the following: given a specific conflict in G (represented by a subgraph G_C of G), find a change of constraint bounds, such that the conflict is removed. More specifically, we are given an instance $\mathcal{I} = (G, w, \tau)$ being a triple of:

1. a PESP instance represented by its constraint graph $G = (V, A)$;
2. a weight vector $w \in \mathbb{R}^{2m}$: for each bound of each constraint a weight is given;
3. a vector $\tau \in \mathbb{R}^{2m}$ denoting by how much each bound of each constraint can be changed;

Among all possible constraint bound changes, we aim to find the one that minimizes a weighted sum of the deviations from the original constraint set (to be defined more specifically in the next paragraph).

In the following paragraphs, we present our approach for solving conflicts in PESP. We present our approach based on (PESP-IP), since this is the most intuitive MIP formulation for PESP. However, the cycle periodicity formulation (CPF) can be extended analogously, and we use an extension of this formulation, (19), for our experiments in Section 4 to speed-up computation times. Details can be found in the Appendix C.

The methods we describe in this Section can be applied to conflicts of any size. In particular, by choosing $G_C := G$ we could consider the whole network as *one* conflict to be resolved.

However, there is a trade-off between computation time for solving individual conflicts: the smaller the conflict, the shorter the time to solve it, and the more constraints contained in each conflict, the less conflicts we have to solve. In particular, in realistic instances as the ones we consider in our experiments in Section 4, G is too big to be resolved at once, which is why we follow the proposed iterative approach of detecting and resolving conflicts one by one. The trade-off between time needed to resolve one conflict and total number of conflicts to be resolved is further discussed in Section 3.2.3. In Section 3.2.4 we propose several ways of enriching conflicts. The described trade-off is illustrated in our experiments in Section 4.

3.2.1 Basic models.

A conflict can be represented by a conflict graph $G_C = (N_C, A_C) \subseteq G$ where N_C and A_C are the nodes (events) and arcs (constraints) in the conflict. Let h_a and t_a be the 'head' and 'tail' of arc

a respectively, i.e. for $a = (i,j) \in A$ we have $h_a = j$ and $t_a = i$. In order to resolve a conflict, bounds have to be changed. To be able to decrease lower bounds and to increase upper bounds, we introduce two new sets of (slack) variables: $s^l_a \in [0, \tau^l_a]$ indicates by how much the lower bound is decreased for a constraint a, and $s^u_a \in [0, \tau^u_a]$ gives by how much its upper bound is increased. Then, for a general graph $G = (V, A)$, and thus specifically for the conflict graph G_C, we obtain the model

$$\text{(PESP-IP-Ext)} \qquad \min \sum_{a\in A} w^l_a s^l_a + w^u_a s^u_a \tag{5a}$$

$$\text{s.t.} \quad l_a - s^l_a \le \pi_{h_a} - \pi_{t_a} + Tp_a \le u_a + s^u_a, \qquad \forall\, a \in A \tag{5b}$$

$$s^l_a \in [0, \tau^l_a],\ s^u_a \in [0, \tau^u_a]. \qquad \forall\, a \in A \tag{5c}$$

$$\pi \in [0, T)^n, \quad p \in \mathbb{Z}^m_+, \tag{5d}$$

where w^l_a and w^u_a are the weights corresponding to the lower and upper bound of constraint a respectively. The choice of weights in such a model depends heavily on the specific application and what kind of solutions are sought [Großmann et al., 2015].

An equivalent formulation based on the cycle periodicity formulation for PESP can be found in Appendix C, see the model in (18).

3.2.2 Fixed trip times.

We now discuss an extension of the described model which includes adjustments to bounds related to fixed trip times.

One advantage of assuming fixed trip times is that the problem size is reduced, only half of the event times need to be determined. Furthermore, in order to model safety constraints on single tracks, a known and fixed trip time is necessary. For more details on how railway-specific requirements can be modeled as PESP constraints, see Appendix A.

Having fixed trip times implies that trip-dwell and trip time constraints are stated between two departure events and incorporates both a trip period and a dwelling period at a station. This dwell time interval is denoted by $[\underline{d}, \overline{d}]$. If the trip time for this part of the network equals r, the trip-dwell constraint is of the form

$$r + \underline{d} \le \pi_{h_a} - \pi_{t_a} + Tp_a \le r + \overline{d}. \tag{6}$$

In the case that dwell time is fixed (i.e. $\underline{d} = \overline{d}$), or that a train does not stop at a station, we have that $l_a = u_a = r + \underline{d}$ for this constraint.

When adjusting constraints involving fixed trip times, we do not relax constraint bounds as in Section 3.2.1, but allow the trip time r itself to change. That means that, when the trip time is changed by ϵ, constraint (6) is rewritten to

$$r + \epsilon + \underline{d} \le \pi_{h_a} - \pi_{t_a} + Tp_a \le r + \epsilon + \overline{d}, \tag{7}$$

rather than widening the span of the constraint.

Trip time changes can imply changes to bounds of safety constraints depending on this trip time, as is illustrated in the following example, and detailed in Appendix A.

Example 3.2. Consider the headway time between trains 1 and 2 entering a station, i.e., the time difference required between the arrival times of the trains in order to safely enter the station. For a headway time of h, the constraint of type (5b) modelling this requirement reads

$$h \le \pi_{a_2} - \pi_{a_1} + Tp \le T - h, \tag{8}$$

where a_i denotes the arrival event of train i. However, since we state constraints in departure events when assuming fixed trip times we use the fact that we have $\pi_{a_i} = \pi_{d_i} + r_i$ (arrival time is departure time plus trip time for train i) to reformulate (8) as

$$h \le \pi_{d_2} + r_2 - \pi_{d_1} - r_1 + Tp \le T - h,$$

or

$$h + r_1 - r_2 \le \pi_{d_2} - \pi_{d_1} + Tp \le T - h + r_1 - r_2.$$

Note that we used this reformulation already in (3h). When the trip times r_1 or r_2 change, this safety constraint should change accordingly.

In order to model these dependencies among constraints, we introduce variables c_a^l and c_a^u to denote the total change made to a constraint $a \in A$. Changes can be made to a constraint because either the constraint *itself* is relaxed, for example by relaxing market requirements, or because of trip time changes in related constraints. So the introduced change-variables consist of two parts: a slack component $s_a \in [0, \tau_a]$, as was introduced in the previous model, accounting for changes in its own constraint bound, and an implied change component c_a^{imp}, accounting for changes to this constraint based on changes in other constraints. This leads to the definition:

$$\begin{cases} c_a^l = s_a^l + c_a^{l,imp} \\ c_a^u = s_a^u + c_a^{u,imp} \end{cases} \qquad \forall\, a \in A.$$

Here, the implied changes can be defined as

$$\begin{cases} c_a^{l,imp} = \left(\lambda_a^l\right)' s \\ c_a^{u,imp} = \left(\lambda_a^u\right)' s \end{cases} \qquad \text{with } \lambda_a^l, \lambda_a^u \in \{0, \pm 1\}^{2m} \qquad \forall\, a \in A,$$

where $s \in \mathbb{R}^{2m}$ is the vector of all slack variables for each constraint and each bound. Here, λ is a vector and the accent denotes it is transposed.

For a detailed explanation of how we model the dependencies of constraints in railway timetabling, see Appendix A.

Summarizing the constraints stated above leads to the mixed integer programming model:

$$\text{(PESP-IP-Dep)} \qquad \min \sum_{a \in A} w_a^l s_a^l + w_a^u s_a^u \tag{9a}$$

$$\text{s.t.} \quad l_a - c_a^l \leq \pi_{h_a} - \pi_{t_a} + T p_a \leq u_a + c_a^u, \quad \forall\, a \in A \tag{9b}$$

$$c_a^l = s_a^l + c_a^{l,imp}, \quad c_a^u = s_a^u + c_a^{u,imp} \qquad \forall\, a \in A, \tag{9c}$$

$$c_a^{l,imp} = \left(\lambda_a^l\right)' s, \quad c_a^{u,imp} = \left(\lambda_a^u\right)' s \qquad \forall\, a \in A,\ \lambda_a^l, \lambda_a^u \in \{0, \pm 1\}^{2m} \tag{9d}$$

$$s_a^l \in [0, \tau_a^l], \quad s_a^u \in [0, \tau_a^u] \qquad \forall\, a \in A, \tag{9e}$$

$$\pi \in [0, T)^n, \quad p \in \mathbb{Z}_+^m, \quad c \in \mathbb{R}^{2m}. \tag{9f}$$

The model presented above is an extension of the model in Section 3.2.1 in the sense that it can deal with changes in (fixed) trip times in the way that is shown in Equation (7) in addition to relaxations. If we take $\lambda_a^l = \lambda_a^u = 0$ for each $a \in A$, we have $c_a^{l,imp} = c_a^{u,imp} = 0$, and we arrive at the (PESP-IP-Ext) model again.

An equivalent formulation based on the cycle periodicity formulation for PESP can be found in Appendix C, specifically in (19).

Redistribute time supplements. As described in the previous sections, the choice of the weights in the objective is very important to find good solutions to PESP instances arising in timetabling problems. Another method that can help to guide the solution towards a specific direction is to redistribute time supplements that are added when designing a timetable. The constraints for this are presented in Appendix B, since it is very railway specific.

3.2.3 Resolving only minimal conflicts.

In this section, we give an example to illustrate why identifying and resolving minimal conflicts within our iterative approach can lead to bad performance. Subsequently, in Section 3.2.4 we describe how to enrich conflicts to overcome this issue.

A minimal conflict in the sense of Definition 3.2 does not necessarily grasp the real-life conflict in full detail. There might be more constraints that are relevant to this conflict, but do not appear in the minimal conflict. In Example 1.2, we can observe that constraints that are not included in the minimal conflict considered, can be relevant for making the PESP instance feasible, which is detailed further in Example 3.3.

Example 3.3. Suppose that there are requirements that require the departure of train 1 at station s to be at .20, and the departure of train 2 at the same station to be at .50. That means constraints (3e) and (3f) change to

$$\pi_1^s - \pi_{ABS} \in [20, 20]_{60} \tag{10a}$$
$$\pi_2^s - \pi_{ABS} \in [50, 50]_{60}. \tag{10b}$$

The model in (3) contains a conflict, which is formed by constraints (3a) - (3d). These constraints form a cycle in the constraint graph as well, as can be seen in Figure 1. Considering this cycle $\mathcal{C}$, the bounds on the multiples of T that have to be in this cycle ($q_{\mathcal{C}}$) can be calculated as in (17), leading to

$$a_{\mathcal{C}} = \left\lceil \frac{7 + 30 - 8 - 30}{60} \right\rceil = 0,$$
$$b_{\mathcal{C}} = \left\lfloor \frac{7 + 30 - 8 - 30}{60} \right\rfloor = -1.$$

Note that $a_{\mathcal{C}} > b_{\mathcal{C}}$, which shows that this cycle forms a conflict.

The following changes to the constraint bounds would resolve the conflict formed by (3a) - (3d):

1. Change (3a) from $[7, 7]_{60}$ to $[8, 8]_{60}$.
2. Or change (3b) from $[8, 8]_{60}$ to $[7, 7]_{60}$.
3. Or change (3c) from $[30, 30]_{60}$ to $[29, 31]_{60}$.
4. Or change (3d) from $[30, 30]_{60}$ to $[29, 31]_{60}$.

However, option 3 does not lead to a feasible model when constraints (10a) and (10b) are taken into account as well, since there is still no solution satisfying all the constraints. This clearly shows that resolving a minimal conflict might not lead to a feasible solution and more constraint must be taken into account.

As is shown by the previous examples, considering only a minimal conflict might not lead to a feasible solution, which is why we add constraints to minimal conflicts in order to capture the real-life conflict better.

Enriching conflicts increases the probability of finding good solutions, since we might have captured the actual conflict better. There are several ways to add more arcs to the conflicts, which is explained in the next subsection.

3.2.4 Methods to enrich a conflict.

In the remainder of the section we call a subgraph of the total timetabling graph containing some conflict a *conflict graph*. If the problem represented by this graph is a minimal conflict, we call it a *minimal conflict graph*. When enriching a conflict, we always start with a minimal conflict.

The models described in the previous section aim at resolving a part of the full PESP network, representing the railway timetabling problem.

In this session we describe various methods to enrich minimal conflicts. First of all, for all trip time and trip-dwell time constraints that are involved in the conflict, we add all constraints that depend on these trip times. For details, see Appendix A.

In the following, we describe how a conflict can be enriched further.

An event in a PESP instance specifies a departure or arrival at some location in the railway network. A conflict, or more generally, a set of events, corresponds to some physical location. This property is used in defining several of the methods listed below.

1. **Use n_1 previous conflicts**
   If hardly any constraints are added to a conflict, it might imply that there still is a conflict around the same location in the PESP network, but now represented by a different set of constraints. Hence, it may happen that in the next step of our iterative approach we find a conflict that is related to the previously found conflict. Therefore, it might be beneficial to resolve these conflicts simultaneously. Also previous conflicts can be taken into account, possibly up to a maximum of n_1 iterations back, where n_1 is a parameter to be chosen.

   In order to define this method, let V_i and V_{i-j} be the set of nodes in the current conflict and the conflict j iterations ago respectively. This method can be executed as follows. Start with $j = 1$ and as long as $j \leq n_1$, check if $V_i \cap V_{i-j} \neq \emptyset$. If this is true, set $V_i = V_i \cup V_{i-j}$, else we stop.

2. **Add neighbouring constraints (depth n_2)**
   As seen in Example 3.3, it can be useful to add more events to a conflict. Therefore, in this method we take all events in the conflict graph and add all incoming and outgoing arcs of the corresponding nodes to the conflict graph, including the corresponding events.

   After having done this addition, it can be repeated several times. The number of times we perform this operation is called the *depth* and is denoted by n_2.

   A practical interpretation of adding neighbouring constraints is that in this way more events of the same train or more events of trains using the same tracks are added.

3. **Add all interrelated constraints**:
   One way to add more useful constraints to a conflict, without including new events, is to add all constraints for which both corresponding events are already involved in the conflict. This rule for enriching a conflict is called *add all interrelated constraints.* This does not enlarge the conflict in the number of events, or in its geographical interpretation, it only adds more constraints.

   If the conflict shown in Example 1.2 were to be enriched by only adding interrelated constraints, no new useful relations would be added. If the incoming and outgoing constraints were added (already for $n_2 = 1$), the conflict would be fully captured and a good solution would be proposed.

4. **Add single track**
   Many conflicts in the rail network arise when single track legs are involved. The best way to resolve conflicts here is to consider all trains sharing this part of infrastructure. Otherwise, there is a high risk of not resolving the conflict in the right way, thus postponing the problem. If we *add single track*, we add all constraints and events that involve trains on this single track leg of the infrastructure.

5. **Add n_3 trips**
   In some cases, it is useful to look at the trains involved in the conflict, and where they travel on the physical rail network. Then the conflict can be enriched by adding n_3 more trips of the trains involved in the conflict (corresponding to n_3 trip(-dwell) constraints). This means the conflict is extended geographically for all involved trains.

   For each train, we add, if possible, n_3 trips before and n_3 trips after the part that is involved in the conflict, if this is possible. This method of adding additional trips is called *add n_3 trips.*

All of these methods can be used on their own to enrich minimal conflicts, or be combined. They are evaluated experimentally in Section 4.

We now describe how we encode the methods we use for enriching conflicts. Note that also the order in which the methods are executed influences the final conflict graph. For ease of notation, we encode a combination of the methods as follows. Each encoding is made in the following way: $n_1 n_2 b_1 b_2 n_3 b_3$, where n_i corresponds to an integer number and b_i to a boolean value ($i \in \{1, 2, 3\}$). All these correspond to the rules described before and are defined as follows:

n_1 This refers to the method *add previous conflicts*, with parameter n_1 (option 1).

n_2 This value denotes the neighbourhood depth for adding neighbouring constraints (option 2).

b_1 If this is *true*, we add all interrelated constraints (option 3).

b_2 If this is *true*, we add single track (option 4).

n_3 We apply the rule ‘add n_3-trips’, i.e., for each train in the conflict, we add n_3 trips before and after the trips involved in the conflict (option 5).

When enriching a conflict, all these methods are executed in the order they appear in the encoding. As an example of an encoding, the code *00ff0f* refers to the method that always resolves a minimal conflict without adding further constraints.

3.3 An iterative algorithm to resolve feasible PESP instances

We now combine what is described in the previous subsections. In Section 3.3.1, we describe the individual iterations of the algorithm. In Section 3.3.2, we use this as a subroutine to describe the final algorithm to resolve all conflicts in a PESP instance.

3.3.1 Resolve a single conflict.

In this section we state Algorithm 3.3 to enrich, and resolve conflicts. A key observation for understanding the design of Algorithm 3.3 is the observation that the computation time for a model like (PESP-IP-Dep) in (9) can increase rapidly as the model size increases, since PESP is NP-complete [Serafini and Ukovich, 1989]. So we have to be careful not to increase computation time too much when adding additional constraints, while still taking care of the solution quality. Adding neighbouring constraints and events is a possibility that can provide useful constraints. However, in dense constraint graphs, this might lead to a large model.

Now suppose a minimal conflict graph is given. Adding more constraints to this graph might make it very large such that it is hard to solve to optimality in reasonable time. Consequently, in Algorithm 3.3, if we cannot resolve the enriched conflict in a certain time limit, we resolve the minimal conflict graph instead, and then successively add constraints. In doing so, we can use the objective value found in the previous step as a lower bound, supported by the following proposition:

Proposition 3.1. Suppose two graphs $G_1 \subseteq G_2$ are given, and the optimal objective values according to the corresponding MIP model (9) are z_1^* and z_2^*. Then $z_1^* \leq z_2^*$.

Proof. Since G_2 contains at least all constraints of G_1, the corresponding model corresponding to G_1 is a relaxation of the model of G_2 and hence $z_1^* \leq z_2^*$. □

Algorithm 3.3 is parametrized by the encoding as described in Section 3.2.4 which defines the *maximum level of additional constraints*. We solve conflicts by solving the corresponding MIP model based on the cycles (19). The subgraph corresponding to the 'maximum level of additional constraints' is denoted by C^+, and its corresponding encoding is denoted by enc^*. Let $\mathcal{S}$ be a list, containing the subgraphs for C^+ that are generated by the encodings from Section 3.2.4 in the following order (where the variable with the asterisk denote the value they have in encoding enc^*):
For $n_2 \in \{0, \dots, n_2^*\}$: for $b_2 \in \{false, b_2^*\}$: for $n_3 \in \{0, \dots, n_3^*\}$: for $b_1 \in \{false, b_1^*\}$: generate the subgraph corresponding to the encoding $n_1^* n_2 b_1 b_2 n_3 b_3^*$ and name them $S_1, \dots, S_\kappa$ in the order they are generated. So $S_i \in \mathcal{S}$ $(i = 1, \dots, \kappa)$ and $\kappa = |S|$. Furthermore, S_1 is the minimal conflict, and $S_\kappa = C^+$. The MIP models, (19) or (9), corresponding to these subproblems can be solved consecutively. Each MIP model corresponding to S_i provides a lower bound for the model corresponding to S_{i+1}, if it is a subgraph, according to Proposition 3.1.

Algorithm 3.3.

```
Data: Conflict $C$, time limits $t_1, t_2, t_3$, weight vector $w \in \mathbb{R}^{2m}$, bound vector $\tau \in \mathbb{R}^{2m}$.
Result: One of the following: 1. Values for $c_a^l$ and $c_a^u$ (for each $a \in A_C$) such that changing the
        constraint bounds by these values leads to a feasible resolution of conflict $C$; 2. Proof that no
        resolution exists; 3. Time-limit exceeded;
1  Enrich conflict $C$, giving enriched conflict $C^+$;
2  Build conflict graph for $C^+$;
3  Solve (CPF-Dep) model for $C^+$ in $t_1$ minutes (dynamic time limit);
4  if optimal solution found to (CPF-Dep) then
5      Process solution found to $C^+$ by changing constraints;
6  else
7      Find sub-problems $S_1, \dots, S_\kappa$;
8      Set $\mathcal{S} \leftarrow S_1$;
9      Solve (CPF-Dep) model for $\mathcal{S}$ in at most $t_2$ minutes (dynamic time limit);
10     for $i = 2, \dots, \kappa$ do
11         Solve $S_i$ in $t_3$ minutes (dynamic time limit). If $b_3$ is set to true, use solution to $S_{i-j}$ as a
             warm start and lower bound, with $j \in \mathbb{N}_{>0}$ as small as possible such that $S_{i-j} \subseteq S_i$;
12         if optimal solution found for $S_i$ then
13             Set $\mathcal{S} \leftarrow S_i$;
14         else
15             Go to line 17;
16         end
17     end
18     Process the solution found to $\mathcal{S}$ by changing constraints, by setting:
19     $l_a \leftarrow l_a - c_a^l$
20     $u_a \leftarrow u_a + c_a^u$ ;
21 end
```

In this algorithm, we take as input a conflict C. Next, we enrich this conflict (line 1). We then try to resolve the conflict in the resulting constraint graph by the (CPF-Dep) model (line 3). We allow t_1 minutes for this. The time limits in the algorithm are dynamic. That means the following: If the time limit is ρ minutes, we allow at least ρ minutes to solve the model. If in the last $\rho/4$ minutes of this allowed time improvements are found, either in the lower bound or the upper bound of the model, we continue optimization for another $\rho/4$ minutes, until no improvements are found any more.

If in these t_1 minutes the model is solved to optimality, we use the found solution as a solution to the conflict (line 5). If not, we start by the smallest subproblem, and solve it (line 9). The time limit t_2 is usually set high in order to ensure that an optimal solution to the minimal conflict is found. This solution is then used as a lower bound for resolving the next conflict graph for which we impose a time limit of t_3 minutes (line 11). If a time limit is exceeded or all subproblems are solved, the last best found solution is used to resolve the conflict (line 17).

If we cannot solve C^+ directly and start solving subgraphs iteratively, these subgraphs are subproblems of each other. Therefore, the objective value of the smaller problem provides a lower bound to the larger subproblem. As soon as we solve such a larger subproblem to resolve a conflict and we know such a lower bound, we provide a constraint to the model stating that the objective should be larger or equal to this lower bound, to help the optimization process. Furthermore, the solution found to a smaller problem can be feasible to the larger problem, thus providing a 'warm start' in the optimization. Whether we supply this solution as a warm start or not is encoded in a Boolean parameter b_3.

3.3.2 Resolve a full PESP instance.

Until now, we have seen how one could resolve a single conflict. Now we describe the final algorithm that, given a PESP instance, iteratively searches for conflicts, resolves them by Algorithm 3.3 (or notifies us that no feasible solution exists) and, if it terminates, provides a conflict-free PESP-instance and a schedule.

Algorithm 3.4.

```
Data: PESP instance $\mathcal{I}$, time limits $t_1, t_2, t_3, TL$, iteration limit $IL$, weight vector $w \in \mathbb{R}^{2m}$, bound
      vector $\tau \in \mathbb{R}^{2m}$.
Result: One of the following: 1. List of adapted PESP constraints such that conflict is resolved
        together with a schedule; 2. Proof that no resolution exists; 3. Time-limit or iteration-limit
        exceeded;
1  Set $it\_count = 0$;
2  Search for a conflict $C$.;
3  while A conflict $C$ is found and $it\_count < IL$ and elapsed time $< TL$ do
4      Run Algorithm 3.3 on conflict $C$;
5      if $C$ cannot be resolved then
6          Stop algorithm: it is not possible to modify PESP parameters within the prespecified bounds to
           obtain a feasible solution to PESP;
7      else
8          Search for a new conflict $C$ in $\mathcal{I}$.;
9      end
10 end
```

This algorithm finds a conflict in the PESP instance or certifies that no such conflict exists.

Next, it resolves the conflict. Afterwards, the search for more conflicts is continued until no conflict exist any more. If there are no more conflicts, we find a schedule and post-optimize it see Section 3.3.4.

Note that if Algorithm 3.3 determines that a certain conflict cannot be resolved, this means that there is no feasible solution to PESP, even with the allowed modifications of the constraint bounds.

Due to the time limits set in line 3, Algorithm 3.4 will terminate after a pre-specified time, even if the conflicts in the network are not resolved up to that point in time and no unresolved conflict is found.

Note that for fixed values of the modulo parameters p, (PESP-IP-Ext) is totally unimodular. Hence when all constraint bounds l_a and u_a and all entries of τ are integer, every time we run Algorithm 3.3 we widen at least one constraint interval by at least one unit. Since we do not allow to undo the changes in step 9, the algorithm terminates at the latest when all PESP constraint bounds are maximally relaxed, that is after $\sum_{a\in A}(\tau_a^l + \tau_a^u)$ iterations.

That implies that if we ran the algorithm without time limits on an an instance of type (PESP-IP-Ext), it would terminate in finite time (under the assumption that RAM and memory of our computer are enough to resolve the occurring conflicts as specified in Algorithm3.3).

For instances of type (PESP-IP-Dep), that is, in particular, when we model changes in the trip times as adjustments of trip times instead of a relaxation of the constraint interval, we can give no such guarantee since a trip time that is changed once from value r to r' may be changed back in a later iteration, if the corresponding constraint occurs in a second conflict. So-induced cycles can be prevented to a certain extent by enlarging conflict graphs as described in Section 3.2.4. A second approach to avoid the repeated changing of trip times is described in the following section.

3.3.3 Tabu search.

In Algorithm 3.4, conflicts are resolved iteratively. If one conflict is resolved, we search for the next one and resolve it. One thing that might happen is that the resolution of one conflict in some iteration, creates a new conflict. If we undo the first change, it might resolve the second conflict but again generate the first one.

There are several possibilities to avoid this going back-and-forth. Note that this situation would have been avoided if more constraints would have been taken into account in resolving the first conflict. However, as this increases the size of the problems to be solved, we use a different approach in addition. For each $a \in A$, we define κ_a as the number of times the slack variables for this constraint have been changed. Then the objective coefficient, penalizing changes to this constraint, is multiplied by g^{κ_a} $(g > 1)$ to make changes to this constraint more expensive and at the same time avoiding models to become infeasible if this constraint has to be changed back again. Throughout the experiments, we use $g = 2$.

3.3.4 Finding a schedule & post-optimisation.

If the iteration limit or time limit in Algorithm 3.4 is not exceeded, the algorithm terminates with a conflict-free PESP instance. A dedicated PESP-solver can be used to solve such an instance (cf. Schrijver and Steenbeek [1993], Kümmling et al. [2015a]). Once we have this schedule, together with the original PESP instance, we can build the (PESP-IP-Dep) model (9) and restrict the event times to be in an interval, given by the event time from the schedule plus or minus a deviation $\eta \in [0, T/2]$, where $\eta = 0$ corresponds to no freedom and $\eta = T/2$ to full freedom. So suppose event π is scheduled to be at .27, we then add the restriction that $\pi \in [27 - \eta, 27 + \eta]$. We call this model PESP-IP-Dep(η). The reason to use a model based on PESP-IP rather than CPF, is that the event times are explicitly included in this model.

If we first solve the model with $\eta = 0$, we find the changes to the constraints that are needed to make the found schedule feasible and we can determine the objective value that is achieved when making this PESP instance feasible. In a next step, we increase η to a positive value and resolve the model. Since the schedule was feasible for PESP-IP-Dep(0), it surely is for PESP-IP-Dep(η) where $\eta > 0$. Hence, better solutions might be found since the feasible space of this model increases in η. So reoptimizing leads to a new change in the constraint bounds and a new objective value that is not higher than the objective value obtained with $\eta = 0$.

4 Experiments

In order to test our methodology described in Algorithm 3.4 in practice, and to analyze the impact of different methods to enrich conflicts, we have tested it on several instances that are provided by Netherlands Railways and are based on the Dutch railway network. They vary between a tiny instance (Section 4.1) and the complete Dutch network (Section 4.3). We present a description of the instances, together with the computational results in this section. Computations are performed on a 64-bit PC with an Intel(R) Core(TM) i7-4700MQ processor with 3.40 GHz and 16.00 GB of RAM installed, operating under Windows 7 Professional. The implementation is done using JAVA 1.8, and IBM ILOG CPLEX 12.6.2 is used to solve the MIP models, allowing for 4 parallel threads.

In Section 3.2.4, several methods are shown to enrich conflicts. Different ways of enriching conflicts can result in different solutions to the model on how to change some of the PESP-constraints. This in turn can influence the number of conflicts found in total. If a solution to some conflict is found that is not only feasible with respect to this specific conflict, but also in the larger context of the total PESP instance, the number of conflicts that have to be resolved is likely to be small, but the computation time might be large. On the other hand, resolving many small conflicts might lead to a lot of iterations of the algorithm, but probably to a relatively short computation time. Therefore, in the experiments we tested many different methods of enriching conflicts, to determine a pattern in which methods work well and which do not.

For finding a conflict or a schedule, the CADANS solver [Schrijver and Steenbeek, 1993, Odijk et al., 2002] is used. This solver finds a minimal conflict or a schedule based on Constraint Programming. For an instance of the size of the Dutch railway network, most of the times computation time is only a few minutes (cf. Caimi et al. [2017, sec. 6.1]) However, our methodology does not depend on the algorithm used for conflict detection and any other algorithm for conflict detection could be used instead.

To solve the conflicts, we used the approach described in Section 3.2.2 since the considered instances contain fixed trip time constraints. Instead of solving (9) however, we used an equivalent formulation based on the cycle periodicity formulation, stated as (19) in Appendix C, which has been proven to be faster in practice. Details on how this formulation is derived can be found in the appendix. To solve (19), a minimum weight spanning tree is found where the weights of the arcs are the number of possibilities for the corresponding process time, i.e., $w_a = u_a - l_a + 1$ for constraint $a \in A$. Based on this tree, an integral cycle basis is generated.

Throughout the post-optimization, we first solve the PESP-IP-Dep(η) model with $\eta = 0$, to determine the initial objective value. Next, we relax the model to $\eta = 5$ and reoptimize, to find the objective value for this model with more freedom.

4.1 Den Helder - Schagen (Hdr-Sgn)

The first instance we consider is a small single-track network between the Dutch cities Den Helder and Schagen.

4.1.1 Instance description.

Physical network. The network consists of four stations as is shown in Figure 3, where the layout of this instance is displayed. Between Anna Paulowna and Den Helder Zuid is a bridge that can be opened during some time of the day and hence no trains can pass during that time. This means that the passing times of trains at this bridge have to be known in order to know when the bridge can be opened. Note that trains can pass each other only at the stations, since the intermediate tracks are all single track.

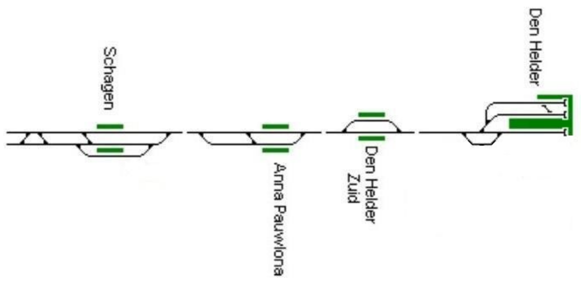


Figure 3: Den Helder - Schagen network. Green parts denote platforms.
Source: www.sporenplan.nl (May 1, 2017).

Train services. On this network, two train lines are operated. Line 1 is an intercity line, stopping only at Den Helder and Schagen and running in both directions, twice per hour. Line 2 is the so-called sprinter line, that runs once per hour and stops at all stations.

Constraints. For service reasons, line 1 is required to drive in a 30-minute pattern, i.e. the trains leave the first station of their journey 30 minutes apart from each other. Furthermore, they should leave Den Helder at .29 or .59, because of connection times to other modes of transport.

In Table 1, an overview is given of the different PESP-constraints that are present in this instance. A trip time constraint refers to a situation where a train does not stop at some station, a trip-dwell constraint refers to the situation where a train drives to a station and dwells there for some time. A synchronisation constraint requires trains to drive in a given pattern (for example, 30 minutes apart). A fixation constraint specifies a specific departure time. Safety constraints all deal with headways.

Objective coefficients and algorithmic limits. A trip(-dwell) constraint involves a given planned trip time r, including a dwell time in case of trip-dwell constraints. This trip time is used to define the maximum allowed changes to the constraint bounds as shown in column 3 and 4. The objective coefficients for the changes are shown in the last two columns. A linear objective is used in this model.

In this instance, we do not allow to change safety constraints, all other constraints can be adjusted. The parameters set were chosen together with NS planners and can be found in Table 1.

As time limits, we used one hour, and the iteration limit was set to 500. Because this instance is small, these limits are actually never met.

4.1.2 Results of the methodology.

When running the algorithm, a minimal conflict was found involving the intercity lines in both directions and the sprinter line in one direction. In total 25 nodes and 38 constraints were involved.

Constraint type	# constraints	Maximum allowed slack		Objective coefficient	
		Lower bound	Upper bound	Lower bound	Upper bound
Trip time	20	$\min\{1, r\}$	2	$\max\{20 - 2r, 10\}$	$\max\{7 - r, 2\}$
Trip-dwell	4	$\min\{1, r\}$	2	$\max\{18 - 2r, 8\}$	$\max\{6 - r, 2\}$
Synchronisation	2	10	10	5	5
Fixation	2	5	5	5	5
Safety	156	0	0	0	0

Table 1: Bounds and coefficients for changes

Using $00ff0f$ as the maximum level of additional constraints, a feasible schedule is found in two iterations.

The proposed solution is to decrease the trip time on one of the trips, i.e., to accelerate the train, and to change the frequency of the intercity line to 29/31 instead of 30/30. The fixations are relaxed by one minute. The total computation time is 0.9 seconds, while finding a solution for the models to resolve a conflict took 0.2 seconds.

The most prominent reason why the resolution of the first conflict caused another conflict, is that the sprinter train was involved in one direction only. If both directions were taken into account, which can be achieved by enriching a conflict by the method $00ft0f$ for example, the solution was found in one iteration. On single track networks, every train that uses this track provides a large limitation on the changes that are feasible. The solution in this case is identical to the solution that was found in the case with the two iterations. Computation times are comparable as the total computation time now is 0.7 seconds and 0.2 seconds are used to resolve the conflict.

4.2 Rotterdam - Utrecht (Rtd-Ut)

4.2.1 Instance description.

Physical network. The underlying network for this instance are the tracks between Gouda and Zwolle. From Gouda, there are also tracks both to Rotterdam and The Hague, and from Zwolle there are tracks both to Groningen and Leeuwarden. The majority of this network is double track, except for the part between Gouda and Utrecht, which has four tracks. The interested reader is referred to www.sporenplan.nl for more details about this network.

Train services. The basis for the instance are the intercity trains that share tracks between Rotterdam and Utrecht. Next to this, we added some additional trains. In total, the instance consists of the following trains (the numbers are added for references later on):

500 Intercity between Rotterdam and Groningen (1 time per hour).

12500 Intercity between Rotterdam and Leeuwarden (1 time per hour).

2000 Intercity between Rotterdam and Utrecht (2 times per hour).

2800 Intercity between The Hague and Utrecht (2 times per hour).

4000 Sprinter train between Rotterdam and Uitgeest (2 times per hour). This train shares infrastructure with the intercity trains between Rotterdam and Gouda. Next, it has to cross the intercity paths halfway between Gouda and Utrecht.

5600 Sprinter train between Utrecht and Zwolle.

Constraints. Lines 2000 and 2800 are synchronized to drive exactly 30 minutes apart from each other. Furthermore, lines 500, 12500 and 2800 are synchronised at Utrecht and Rotterdam to drive 15 minutes apart from each other.

Objective coefficients and algorithmic limits. In Table 2 the maximum allowed changes and the objective coefficients are shown, which were chosen based on expertise from NS planners. Furthermore, the number of constraints of each type is shown.

The PESP instance contains 699 events and 3061 constraints. This instance contains a lot of intercity trains, sharing the same infrastructure. However, all these trains might have different driving characteristics as they use different train types. Therefore, the corresponding trip times

Constraint type	# constraints	Maximum allowed slack		Objective coefficient	
		Lower bound	Upper bound	Lower bound	Upper bound
Trip time	512	$\min\{1,t\}$	2	$\max\{20-2t,10\}$	$\max\{7-t,2\}$
Trip-dwell	167	$\min\{1,t\}$	2	$\max\{18-2t,8\}$	$\max\{6-t,2\}$
Frequency	93	10	10	30	30
Fixation	18	5	5	20	20
Connection	14	1	5	30	25
Safety	2257	0	0	0	0

Table 2: Bounds and coefficients for changes in Rtd-Ut

can be different, which is undesirable because it consumes more capacity on these tracks and it easily gives rise to conflicts. Hence, solutions that redistribute buffer times (see Appendix B) are desired. In order to find these, the corresponding constraints are added and the corresponding objective term gets a coefficient of value 10 (the γ in (16)).

For the algorithm, we set an iteration limit of 500 iterations and a time limit of 2 hours.

4.2.2 Results of the methodology.

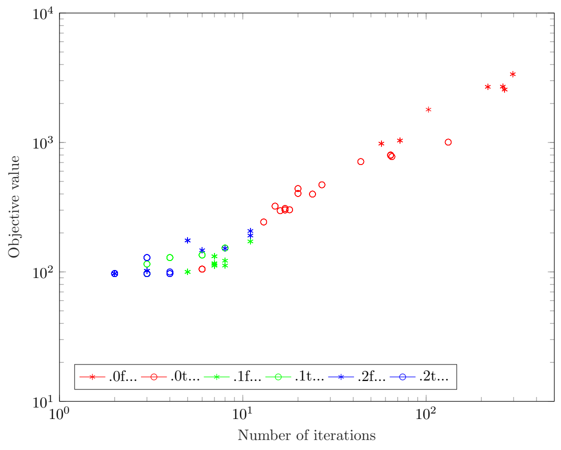
Using the encoding stated in Section 3.2.4, we tested all possible combinations of the following ways to enrichting a conflict:

- $n_1 \in \{0,1,2\}$, the number of previously found conflict that are added.
- $n_2 \in \{0,1,2\}$, the neighbourhood depth.
- $n_3 \in \{0,2,4,6,8,10\}$, the number of additional trips of the train lines that are involved.

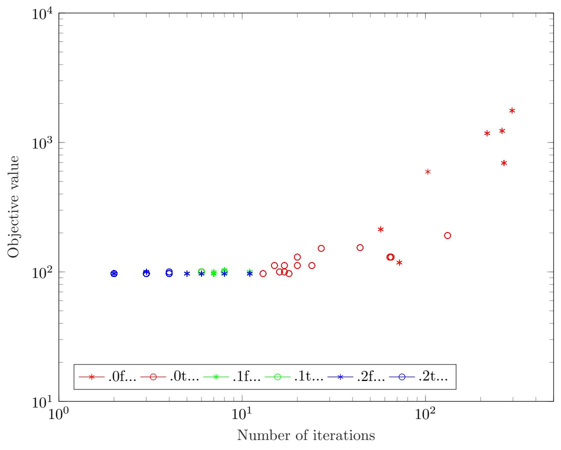
Furthermore, the true/false parameters could take both values, which correspond to adding the interrelated constraints or not, adding all trains in single track parts or not and fixing the process times of constraints when generating lower bounds.

This led to 432 possible methods of adding additional constraints. Out of these 432 methods, 48 have led to the algorithm exceeding the iteration limit. All of these 48 methods had neighbourhood depth equal to zero and no interrelated arcs were added.

For the remaining methods, results are shown in Figures 4 and 5. The legend shows which method is used to enrich conflicts. For example, *.1t...* shows that first all constraints are added that have a relation with the conflict, and next all interrelated constraints are added. The dots indicate that the parameter for this method is left open, so in the figure, there are several dots corresponding to *.1t...*, all indicating a different setting where the parameter on the dots is varied.



(a) Initial objective

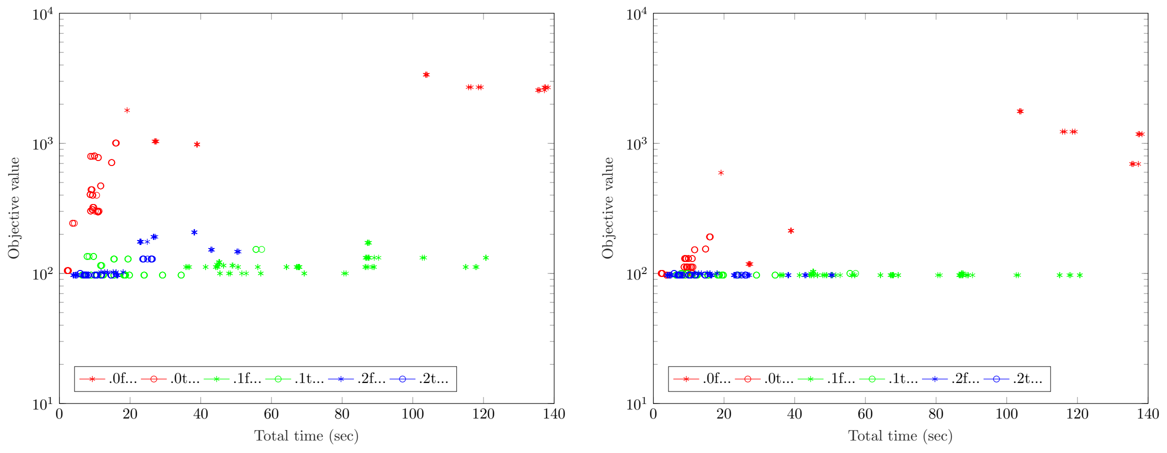


(b) Improved objective

Figure 4: Number of iterations versus objective

In Figure 4, a plot is shown of the number of iterations that are performed versus the total objective value (Figure 4(a)) and the objective value obtained after post-optimization (Figure 4(b)) as is introduced in Section 3.3.4.

By using the three different colors, we distinguish between the results for adding neighbouring constraints or not. Also, we distinguished between adding interrelated constraints or not (option 3 in Section 3.2.4) by using different symbols. As is clear from these figures, adding more constraints leads to a smaller number of iterations and a better initial objective. Also, for the objective obtained after post-optimization, the same observation can be made. For neighbourhood depth 1 and 2, a solution with objective value 97 was obtained in nearly all methods. Also, a difference can be seen between the methods that add interrelated constraints or not. In general, doing so leads to fewer iterations to find a feasible schedule.



(a) Initial objective

(b) Improved objective

Figure 5: Total time versus objective

In Figure 5, plots are shown for the total time in seconds versus the objective. The same color schemes are used here as in the previous figure. As is clear, in general the red dots (neighbourhood depth 0) correspond to methods that lead to solutions being found rapidly, however, with a high objective value in general. Also, if these solutions are improved, they are still worse than the solutions corresponding to methods with a different neighbourhood depth. There are a few exceptions, having a really long computation time and a high objective value.

In general we can observe that resolving only the minimal conflicts does not work well. All the methods that have led to exceeding the iteration limit, hardly added any constraints to the minimal conflict. Furthermore, we saw a tendency that increasing the neighbourhood depth leads to better solutions. However, this may come at the cost of an increase in computation time. A trade-off has to be made, where to stop adding additional constraints. Adding too many leads to an increased problem size, without providing additional relevant information.

4.3 Dutch network 2013 (NL2013)

4.3.1 Instance description.

This instance was used as a basis to generate the full Dutch schedule of 2013, including transfer and synchronisation requirements. The timetabling instance is initially infeasible. In order to obtain the actual schedule in 2013 in the Netherlands, trip time and synchronisation requirements were changed manually.

The Dutch railway network is one of the most heavily used railway networks in Europe [Boston Consulting Group, 2015]. There are many trains operated on this network, many of them sharing a piece of infrastructure in the network, thus generating many interdependencies in the network. Therefore, finding a feasible schedule for this instance is a challenging task. This is why we ran experiments on this network, to test the performance of our algorithmic framework.

The only expected changes are on trip times and synchronisation requirements.

Objective coefficients and algorithmic limits. The allowed changes, the corresponding coefficients and the number of PESP-constraints for each type are shown in Table 3. They are similar to the previous case. Here, the constraints to redistribute buffer times between stations are not added to the IP. We have set a maximum of 500 iterations and 3 hours of computation time. The network consists of 9085 nodes and 75309 constraints with 448 trains in total.

Constraint type	# constraints	Maximum allowed slack		Objective coefficient	
		Lower bound	Upper bound	Lower bound	Upper bound
Trip time	5670	$\min\{1,t\}$	2	$\max\{20-2t,10\}$	$\max\{7-t,2\}$
Trip-dwell	2966	$\min\{1,t\}$	2	$\max\{18-2t,8\}$	$\max\{6-t,2\}$
Frequency	291	10	10	30	30
Fixation	18	5	5	20	20
Connection	0	1	5	30	25
Other	66212	0	0	0	0

Table 3: Bounds and coefficients for changes in NL2013

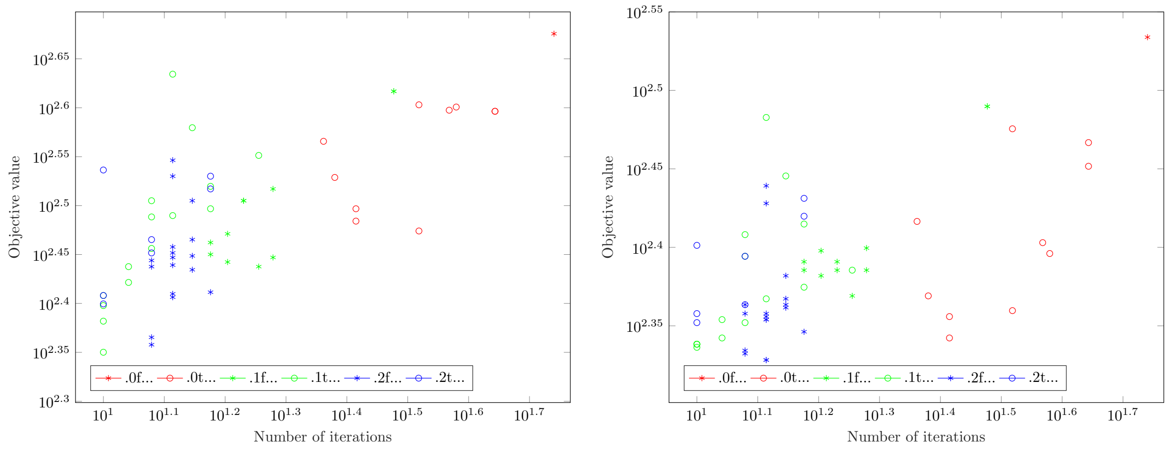
4.3.2 Results of the methodology.

For this instance again several methods are tested to enrich a conflict. We varied the parameters of the methods as follows:

- $n_1 \in \{0,1,2\}$, the number of previously found conflict that are added.
- $n_2 \in \{0,1,2\}$, the neighbourhood depth.
- $n_3 \in \{0,2,4\}$, the number of additional trips of the train lines that are involved.

Furthermore, the true/false parameters could take both values, which correspond to adding the interrelated constraints or not, adding all trains in single track parts or not, and fixing the process times of constraints when generating lower bounds.

This led to 216 possible methods of adding additional constraints. Out of these 216 methods, 1.8% exceeded the iteration limit, which are methods that had neighbourhood depth 0 and no interrelated constraints added. 49.5% exceeded the overall time limit and 20.37% exceeded the time limit allowed for finding conflicts or a schedule in one of the iterations. We do not recognize a clear pattern in which combination of parameteres causes violation of the time limit. The results for the remaining methods are shown in Figures 6 and 7.

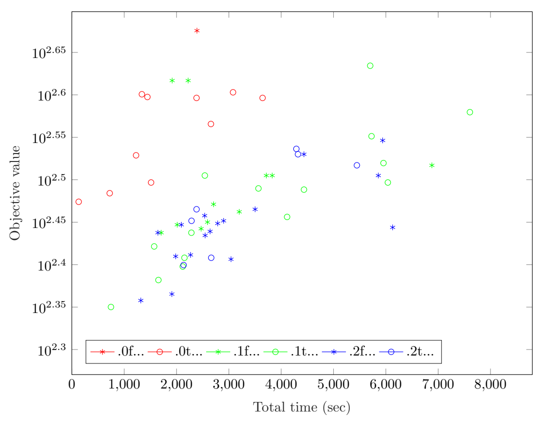


(a) Initial objective

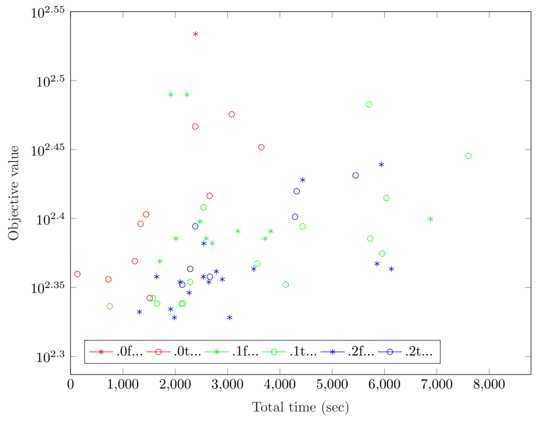
(b) Improved objective

Figure 6: Number of iterations versus objective

Although the results are less clear than in the previous case, again we see that the methods that have neighbourhood depth 0 tend to lead to a higher number of iterations. Although the



(a) Initial objective



(b) Improved objective

Figure 7: Total time versus objective

computation time in general is low, again the objective value is very high. For neighbourhood depth 1 and 2 the results are identical.

Another observation here is that almost all methods that have neighbourhood depth 0 and no addition of interrelated arcs, did not lead to a feasible result within the given time and iteration limits.

5 Conclusion and discussion

We developed a methodology to relax PESP constraints to resolve infeasible PESP instances. This approach supplements current timetabling algorithms, which suffer from the fact that increased demand for capacity usage as well as quality requirements often lead to (on first sight) infeasible timetabling instances. We resolve conflicts in a PESP model with as few deviations as possible, based on predefined constraint weights and bounds. Our approach is iterative in the sense that we find a conflict in the existing PESP instance, solve this conflict using a MIP model, and then search for the next conflict. In contrast to existing approaches on resolving infeasible PESP instances, our approach can deal with fixed trip times, an assumption that is often made in PESP instances arising from railway timetabling.

We find that in our iterative approach of finding and resolving conflicts it is important not to resolve only minimal conflicts, but to carefully add more constraints from the timetabling instance to these minimal conflicts, in order to find good solutions as well as to improve computation times. We proposed several methods to enrich the minimal conflicts. From our computational results, we have seen that adding neighbouring constraints in the constraint graph in general leads to better results.

In our experiments based on parts of or the whole the Dutch railway network, feasible timetables are found in reasonable time in most cases.

Our approach requires the choice of many parameters, namely the allowed deviations of the original bounds and the weights in the objective function, which will steer the algorithm towards a solution. In practice, these parameters would need to be chosen based on the expertise and preferences of the railway operator - and could be adjusted based on the feedback of the operator when seeing the generated solution. It would even be possible to incorporate expert feedback (in the sense of resetting and fine-tuning parameters) after the resolution of each conflict.

Acknowledgements We wish to thank Netherlands Railways for providing the data and access to software systems to carry out this research.

A Several types of PESP constraints

In this appendix, we describe several constraints that arise when modeling railway timetabling instance with known and fixed trip times as PESP. We show that the bounds of many of the safety constraints, depend on some trip time of a train on a part of the railway infrastructure. This is important for the models (19) and (9). More specifically, these dependencies specify the relations in constraints (9d) and (19f). An overview on how to derive the constraints is shown in Peeters [2003].

When two trains share the same piece of infrastructure, we introduce constraints to prevent them from using it simultaneously. Trains can be driving in the same direction or in opposite directions on that infrastructure. The model that is proposed in this paper is a macroscopic model and hence does not focus on block section levels. Instead, headway constraints are used to separate pairs of trains in time.

For notational convenience, let h_a and t_a be the 'head' and 'tail' of an arc in the PESP-graph respectively. Since trip times are assumed to be fixed, one could think of the nodes in a PESP instance as a combination of a departure event, a trip time and an arrival event, aggregated into one contracted node. In this context, trip time constraints link consecutive departures of a train line to each other. The bounds in these constraints consist of the trip time from a departure to the next arrival, and some possible dwell time at the arrival station. If this dwell time should be between $\underline{d}$ and $\overline{d}$ and the trip time is denoted by r, the trip/dwell time constraints are of the form

$$r + \underline{d} \leq \pi_j - \pi_i + Tp_{ij} \leq r + \overline{d}.$$

Other constraints that arise in timetabling are synchronisation constraints ((3c) and (3d) as shown in Example 1.2), or fixations (see (3e) and (3f) in the same example). The majority of the constraints however are on safety, as becomes clear from Tables 1, 2 and 3. In the remainder of this section, the safety constraints are explained, as well as the way they depend on trip times. Also the relationship of connection constraints and trip times is described.

Since trip times are assumed to be fixed, a node in a PESP constraint graph corresponds to a departure from some station, inclusing the trip to the next station. For almost every node, unless it is the last node of a train, there exists an outgoing trip arc. For notational ease, let $e : V \to A$ be the mapping from a node to its outgoing trip arc.

A.1 Trains running in the same direction

When multiple trains use the same infrastructure, safety headways are imposed, that is, to separate the usage of the infrastructure in time and thus prevent conflicts in infrastructure utilization. In reality, such headways are imposed at stations as well as in between stations at any point where a conflict could occur, e.g., at points where train paths cross or merge. For the sake of simplicity, we only refer to stations in the description here. Suppose two trains share the same track between stations 1 and 2 in the network. Furthermore, assume the trains have to be separated in time upon arrival and departure by at least κ minutes. The departure and arrival times of train i at station s are denoted by $\pi^s_{d_i}$ and $\pi^s_{a_i}$ respectively. The travel time between station s and s' for train i is denoted by r_i. If one train departs, the other cannot depart in the κ minutes before or after this departure. This leads to

$$\pi^s_{d_2} - \pi^s_{d_1} \notin (-\kappa, \kappa).$$

Since the schedule is cyclic, we can use this to model the above as a PESP constraint:

$$\pi^s_{d_2} - \pi^s_{d_1} \in [\kappa, T - \kappa]_T,$$

or equivalently by introducing the integer variable p:

$$\kappa \leq \pi^s_{d_2} - \pi^s_{d_1} + pT \leq T - \kappa.$$

This constraint is required for each pair of trains departing from a station s.

For safety upon arrival at station s', a similar constraint holds. If arrival events would be used, the constraint would look like

$$\kappa \leq \pi^{s'}_{a_2} - \pi^{s'}_{a_1} \leq T - \kappa. \tag{11}$$

In order to state this constraint solely in departure events, note that $\pi^{s}_{d_i} + r_i = \pi^{s'}_{d_i}$, i.e., arrival time equals the departure time plus trip time. Substituting this into (11) leads to

$$\kappa \leq \pi^{s}_{d_2} + r_2 - \pi^{s}_{d_1} - r_1 \leq T - \kappa.$$

Rewriting this by moving the trip times to the constraints bounds gives

$$\kappa + r_1 - r_2 \leq \pi^{s}_{d_1} - \pi^{s}_{d_2} \leq T - \kappa + r_1 - r_2. \tag{12}$$

Clearly, if the trip time constraints for which the trip times are involved in (12) change, the bounds of this constraint change. If r_1 or r_2 is changed, it is because $e(t_a)$ or $e(h_a)$ has changed respectively. Note that the increase in r_1 is given by $s^{u}_{e(t_a)} - s^{l}_{e(t_a)}$, and the increase in r_2 is given by $s^{u}_{e(h_a)} - s^{l}_{e(h_a)}$. So the increase in the lower bound for a safety constraint like (12) is given by

$$\left(s^{u}_{e(t_a)} - s^{l}_{e(t_a)}\right) - \left(s^{u}_{e(h_a)} - s^{l}_{e(h_a)}\right),$$

and for the upper bound it is the same. Since the sign of the change variables c^{l}_{a} in (9b) and (19d) is negative and for c^{u}_{a} it is positive, this leads to

$$c^{l,imp}_{a} = -\left(s^{u}_{e(t_a)} - s^{l}_{e(t_a)}\right) + \left(s^{u}_{e(h_a)} - s^{l}_{e(h_a)}\right)$$

and

$$c^{u,imp}_{a} = \left(s^{u}_{e(t_a)} - s^{l}_{e(t_a)}\right) - \left(s^{u}_{e(h_a)} - s^{l}_{e(h_a)}\right).$$

Note that this defines the λ-vectors in equations (9d) and (19f). Here, the s-variables are the slack variables of a (trip time) constraint that $e(h_a)$ and $e(t_a)$ refer to.

A.2 Single track headways

On some part of the rail network, trains use a part of a track in different directions. A train in some direction can only start once the track is unoccupied and the train in the opposite direction has left. There are several constraints that guarantee safety here. These are also used if the train paths of one incoming and one outgoing train cross around a station.

Suppose stations s and s' are given with a single track in between. Trains can only pass at the stations. Train 1 drives from s to s' in r_1 minutes, train 2 drives the other way in r_2 minutes. In Figure 8 a sketch is given in a time space diagram, with time on the horizontal axis and space on the vertical axis. Train 1 is shown twice, the second one means the same train in the next cycle period. Trains have to be separated in time. How much this headway time κ is, can be different if it concerns the time between an incoming train and the next outgoing train (an *in-out* relation, indicated by 'io'), or the other way round (an *out-in* relation, indicated by 'oi'). They can also be station-dependent.

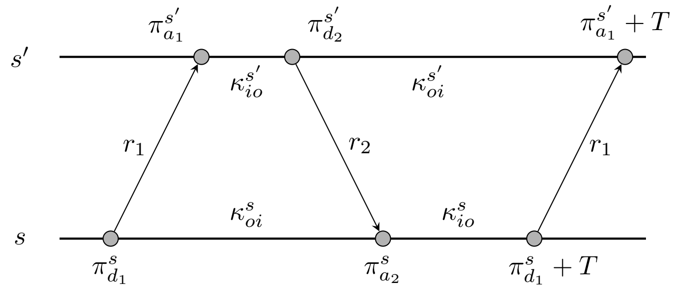


Figure 8: Single track time-space diagram

The constraint to avoid a collision on such a single track is given by

$$\kappa^{s'}_{io} + r_1 \leq \pi^{s'}_{d_2} - \pi^{s}_{d_1} + Tp \leq T - \kappa^{s}_{io} - r_2. \tag{13}$$

The lower bound comes from the fact that $\pi^{s'}_{d_2} - \pi^{s'}_{a_1} \geq \kappa^{s'}_{io}$, which implies

$$\kappa^{s'}_{io} + r_1 \leq \pi^{s'}_{d_2} - \pi^{s}_{d_1}. \tag{14}$$

The upper bound comes from the fact that $\pi^s_{d_1} - \pi^s_{a_2} \geq \kappa^s_{io}$, which implies

$$\pi^s_{d_1} - \pi^{s'}_{d_2} \geq \kappa^s_{io} + r_2 \quad \implies \quad \pi^{s'}_{d_2} - \pi^s_{d_1} \leq -\kappa^s_{io} - r_2.$$

This, by combining with (14) and using cyclicity, leads to (13).

Clearly, trip times are involved in the bounds. Similar to what is shown above, the implied changes to this constraint can be found, due to changes in the trip times. The lower bound depends positively on r_1, hence we have

$$c^{l,imp}_a = -\left(s^u_{e(t_a)} - s^l_{e(t_a)}\right).$$

The lower bound depends negatively on r_2, which leads to

$$c^{u,imp}_a = -\left(s^u_{e(h_a)} - s^l_{e(h_a)}\right).$$

A.3 Crossing train paths

In constraint (13) it is assumed that both trains share the whole single track part. It is possible that trains only use the same infrastructure close to a station and then go different ways. This basically means that no train can leave around the time an incoming train enters the station. The constraint does not have to be as strict as the single-track constraint that prevents the trains meeting on the whole single track part.

A.3.1 In-out relations.

The constraint dealing with the time between an incoming train (train 1) and the next outgoing train (train 2) at station s' is (see also Figure 8)

$$\kappa^{s'}_{io} \leq \pi^{s'}_{d_2} - \pi^{s'}_{a_1} + pT \leq T - \kappa^{s'}_{oi},$$

which becomes (using the fixed trip times)

$$\kappa^{s'}_{io} + r_1 \leq \pi^{s'}_{d_2} - \pi^s_{d_1} + pT \leq T - \kappa^{s'}_{oi} + r_1.$$

The implied changes for the lower bound are (similar to the cases above) given by

$$c^{l,imp}_a = -\left(s^u_{e(t_a)} - s^l_{e(t_a)}\right), \tag{15}$$

and for the upper bound given by

$$c^{u,imp}_a = \left(s^u_{e(t_a)} - s^l_{e(t_a)}\right).$$

A.3.2 Out-in relations.

The previous constraint considers an *in-out* relation. The *out-in* relation leads to the constraint

$$\kappa^{s'}_{oi} \leq \pi^{s'}_{a_1} - \pi^{s'}_{d_2} + pT \leq T - \kappa^{s'}_{io},$$

which is the same as

$$\kappa^{s'}_{oi} - r_1 \leq \pi^s_{d_1} - \pi^{s'}_{d_2} + pT \leq T - \kappa^{s'}_{io} - r_1.$$

Hence, the implied change now are

$$c^{l,imp}_a = \left(s^u_{e(h_a)} - s^l_{e(h_a)}\right)$$

and

$$c^{u,imp}_a = -\left(s^u_{e(h_a)} - s^l_{e(h_a)}\right).$$

A.4 Connections

A general connection constraint states that the arrival event of an incoming train and the departure of another train should be within a certain (small) interval. Suppose that train 1 is the incoming train (trip time r_1) and train 2 is outgoing. This leads to (omitting the irrelevant station index here)

$$\underline{c} \leq \pi_{d_2} - \pi_{a_1} + pT \leq \overline{c}.$$

This is equivalent to

$$\underline{c} + r_1 \leq \pi_{d_2} - \pi_{d_1} + pT \leq \overline{c} + r_1.$$

In this case the implied changes are given by (see also (15)):

$$c_a^{l,imp} = -\left(s^u_{e(t_a)} - s^l_{e(t_a)}\right)$$

and

$$c_a^{u,imp} = \left(s^u_{e(t_a)} - s^l_{e(t_a)}\right).$$

B Redistribution of time supplements

In this appendix, we describe one more set of constraints which can help to guide the solution towards a specific direction.

When designing a schedule, planners take into account that in a daily practice operations do not go as planned. Especially in train scheduling, this is very important, since the vehicles are bound to the tracks, and hence disturbances can easily influence the daily operations, leading to trains being delayed. In order to avoid the disruption of the daily operations as much as possible, schedulers add time supplements to the travel time on some parts of the train journey. That means, they plan that the train uses more time than actually needed, in general between 5% and 7% on the total journey. These additional minutes that are added are called *time supplements*. Within a journey, the schedules add these supplements are added to parts of the journey rather arbitrarily. In this paragraph, we propose a method that finds solutions that redistribute these supplements to different trips, in order to obtain a feasible schedule. This does not change the total journey time of a train, but only redistributes the supplements.

Guiding the search for a solution towards these solutions is modelled by defining a variable $z_a = s^u_a - s^l_a$ for each constraint $a \in A_{trip}$, accounting for the change in trip time. Here, $A_{trip} \subseteq A$ is the set of trip time constraints. Let $T_a \subseteq A_{trip}$ be the set of trip time constraints between the two main stations closest to the part that is involved by constraint a. Then the total change C_a on this part of the train series is calculated as

$$C_a = \sum_{e \in T_a} z_e.$$

If nothing is changed in the trip time between these two main stations, we have $C_a = 0$. Also if the time supplements are redistributed between the stations, this value will be zero as well, since the changes to individual constraints cancel out.

In order to find solutions that redistribute time supplements, we penalize solutions in the objective functions, that do not do this. In order to do so, note that T_a consists of a set of consecutive trip time constraints. Let $a^* \in T_a$ be the first of these consecutive constraints. Next, we introduce a variable $t_a \geq 0$ for each $a \in A_{trip}$ and add

$$\gamma \cdot \sum_{a \in A_{trip}} t_a \tag{16}$$

to the objective, where γ is an objective coefficient to be chosen. Furthermore, we add the constraints

$$-t_{a^*} \leq C_a \text{ and } C_a \leq t^{a^*} \qquad \forall\, a \in A_{trip},$$

to the model, where $a^* \in T_a$ is as defined before. By doing this, we penalize the changes only once. By adding the above objective term and constraints, changes in trip times that are not compensated for in other trip time constraints, get an additional penalty.

C Solving conflicts by extending the cycle periodicity formulation

Analogously to the described extension of (PESP-IP), we can extend the cycle periodicity formulation for PESP to to resolve conflicts.

C.1 The Cycle Periodicity Formulation of PESP.

We now describe a second formulation of (PESP) as integer program [Nachtigall, 1999]. We introduce the notion of a ‘process time’ of a constraint, which is the difference between the event times, i.e., for constraint $a = (i, j) \in A$, the process time x_a is defined as

$$x_a = \pi_j - \pi_i + Tp_{ij}.$$

For connection constraints for example, the process time specifies how many minutes of transfer time are available. It is easy to see that the sum of all process times in a cycle in the constraint graph representation of the PESP instance should be an integer multiple of T. If we denote a cycle in the constraint graph by $\mathcal{C}$, the sum of the process times in this cycle equals $q_{\mathcal{C}}T$, i.e., $q_{\mathcal{C}}$ denotes the number of multiples of T this cycle. Next, if we choose a direction in the cycle $\mathcal{C}$ in which the cycle is traversed, and denote the set of forward and backward arcs of this cycle by $\mathcal{C}^+$ and $\mathcal{C}^-$ respectively, the Cycle Periodicity Formulation is given as follows:

Definition C.1 (CPF)**.** *Given a constraint graph $G = (V, A)$ as defined before, find x_a for all $a \in A$ such that*

$$\begin{aligned}
&\sum_{a\in\mathcal{C}^+} x_a - \sum_{a\in\mathcal{C}^-} x_a = Tq_{\mathcal{C}} && \forall\, \mathcal{C} \in G\\
&l_a \le x_a \le u_a && \forall\, a \in A\\
&a_{\mathcal{C}} \le q_{\mathcal{C}} \le b_{\mathcal{C}} && \forall\, \mathcal{C} \in G\\
&x \in \mathbb{R}^m, \quad q \in \mathbb{Z}^k,
\end{aligned}$$

where k is the number of cycles in the graph and

$$a_{\mathcal{C}} = \left\lceil \frac{1}{T}\left(\sum_{a\in\mathcal{C}^+} l_a - \sum_{a\in\mathcal{C}^-} u_a\right)\right\rceil, \qquad b_{\mathcal{C}} = \left\lfloor \frac{1}{T}\left(\sum_{a\in\mathcal{C}^+} u_a - \sum_{a\in\mathcal{C}^-} l_a\right)\right\rfloor. \tag{17}$$

Although there is an exponential number of cycles in the graph, it is sufficient to require the cycle constraints only for cycles in an integral cycle basis $\mathcal{B}$, i.e., a set $\mathcal{B}$ such that every non-basis cycle is an integer linear combination of the cycles in $\mathcal{B}$ [Peeters, 2003, Liebchen, 2003]. Such a cycle basis can for example be found by first finding a spanning tree in the graph. All the arcs that are not in the tree provide a cycle in the graph, all of them together lead to an integral cycle basis of size $k = m - n + 1$, hence we need only a limited amount of cycles and therefore a limited number of integer variables [Liebchen, 2003].

An overview of the theory concerning cycle bases is given in Liebchen and Peeters [2009].

An advantage of this model is that it uses fewer integer variables and equality constraints instead of inequality constraints. This has advantages in a branch-and-bound procedure when solving the models. For a further discussion and comparison of different PESP formulations, see for example Liebchen et al. [2008].

C.2 Equivalent for (PESP-IP-Ext)

We can extend (CPF) in the same way as described in Section 3.2.1 for (PESP-IP) to resolve conflicts.

We assume a cycle basis $\mathcal{B}$ is given. The Cycle Periodicity Formulation equivalent to (PESP-IP-Ext) can be stated as follows:

$$
\begin{aligned}
\text{(CPF-Ext)} \qquad & \min \quad \sum_{a\in A} w_a^l s_a^l + w_a^u s_a^u && && \text{(18a)}\\
& \text{s.t.} \quad \sum_{a\in\mathcal{C}+} x_a - \sum_{x\in\mathcal{C}-} x_a = Tq_{\mathcal{C}} && \forall \mathcal{C}\in\mathcal{B} && \text{(18b)}\\
& \qquad l_a - s_a^l \le x_a \le u_a + s_a^u, && \forall\, a\in A && \text{(18c)}\\
& \qquad a_{\mathcal{C}} \le q_{\mathcal{C}} \le b_{\mathcal{C}} && \forall \mathcal{C}\in\mathcal{B} && \text{(18d)}\\
& \qquad s_a^l \in [0,\tau_a^l],\ s_a^u\in[0,\tau_a^u]. && \forall\, a\in A && \text{(18e)}\\
& \qquad x\in\mathbb{R}^m, \quad q\in\mathbb{Z}^{|\mathcal{B}|}. && && \text{(18f)}
\end{aligned}
$$

The objective function should be the same and the constraint bounds are altered in the same way, by adding and subtracting the right slack variables, which are the same as well, so Constraint (18a), (18c), (18e) and (18f) are essentially the same. The sum of all process times should still be equal to a multiple of T as in Constraint (18b). Finally, the number of multiples of T in each cycle $\mathcal{C}$ can be bounded by

$$
a_{\mathcal{C}} = \left\lceil \frac{1}{T}\left(\sum_{a\in\mathcal{C}+}(l_a-\tau_a^l) - \sum_{a\in\mathcal{C}-}(u_a+\tau_a^u)\right)\right\rceil, \qquad b_{\mathcal{C}} = \left\lfloor \frac{1}{T}\left(\sum_{a\in\mathcal{C}+}(u_a-\tau_a^u) - \sum_{a\in\mathcal{C}-}(l_a-\tau_a^l)\right)\right\rfloor,
$$

where we assume worst case situations, i.e., we calculate the lowest and highest possible values, based on the maximum allow changes, in order to have a model that is not too restrictive. This clearly shows that the (PESP-IP-Ext) and (CPF-Ext) models are equivalent.

C.3 Equivalent for (PESP-IP-Dep)

In order to get the CPF equivalent for (PESP-IP-Dep), we use a similar approach. The model is stated as

$$
\begin{aligned}
\text{(CPF-Dep)} \qquad & \min \quad \sum_{a\in A} w_a^l s_a^l + w_a^u s_a^u && && \text{(19a)}\\
& \text{s.t.} \quad \sum_{a\in\mathcal{C}+} x_a - \sum_{a\in\mathcal{C}-} x_a = Tq_{\mathcal{C}}, && \forall \mathcal{C}\in\mathcal{B} && \text{(19b)}\\
& \qquad a_{\mathcal{C}} \le q_{\mathcal{C}} \le b_{\mathcal{C}} && \forall \mathcal{C}\in\mathcal{B} && \text{(19c)}\\
& \qquad l_a - c_a^l \le x_a \le u_a + c_a^u && \forall\, a\in A && \text{(19d)}\\
& \qquad c_a^l = s_a^l + c_a^{l,imp}, \quad c_a^u = s_a^u + c_a^{u,imp} && \forall\, a\in A, && \text{(19e)}\\
& \qquad c_a^{l,imp} = \left(\lambda_a^l\right)' s, \quad c_a^{u,imp} = \left(\lambda_a^u\right)' s && \forall\, a\in A,\ \lambda_a^l,\lambda_a^u\in\{0,\pm 1\}^{2m} && \text{(19f)}\\
& \qquad L_a = l_a - \tau_a^l - \max\{c_a^{l,imp}\} && \forall\, a\in A && \text{(19g)}\\
& \qquad U_a = u_a + \tau_a^u + \max\{c_a^{u,impl}\} && \forall\, a\in A && \text{(19h)}\\
& \qquad a_{\mathcal{C}} = \left\lceil \frac{1}{T}\left(\sum_{a\in\mathcal{C}+} L_a - \sum_{a\in\mathcal{C}-} U_a\right)\right\rceil && \forall \mathcal{C}\in\mathcal{B} && \text{(19i)}\\
& \qquad b_{\mathcal{C}} = \left\lfloor \frac{1}{T}\left(\sum_{a\in\mathcal{C}+} U_a - \sum_{a\in\mathcal{C}-} L_a\right)\right\rfloor && \forall \mathcal{C}\in\mathcal{B} && \text{(19j)}\\
& \qquad s_a^l \in [0,\tau_a^l], \quad s_a^u\in[0,\tau_a^u] && \forall\, a\in A, && \text{(19k)}\\
& \qquad x\in\mathbb{R}^m, \quad q\in\mathbb{Z}^{m-n+1}, \quad c\in\mathbb{R}^{2m}. && && \text{(19l)}
\end{aligned}
$$

Here, constraints (19b) relates the processes in a cycle to an integer multiple of T. Constraints (19d) give bounds on the allowed process times and correspond to the bounds of a PESP constraint. Constraints (19c) bound the integer variable for each cycle, denoting how many multiples of T can possibly occur in a cycle. Bounds on these variables are calculated by (19i) and (19j). It makes use of the smallest and largest possible bounds for each constraint, which are calculated by (19g) and (19h), in order to avoid having a model that is too restrictive.

Note that again (PESP-IP-Dep) (9) and (CPF-Dep) (19) are equivalent models.

References

Boston Consulting Group. The 2015 European railway performance index, exploring the link between performance and public cost, May 2015.

Gabrio Caimi, Leo Kroon, and Christian Liebchen. Models for railway timetable optimization: Applicability and applications in practice. *Journal of Rail Transport Planning & Management*, 6(4):285 – 312, 2017. ISSN 2210-9706. doi: http://dx.doi.org/10.1016/j.jrtpm.2016.11.002.

Peter Großmann, Steffen Hölldobler, Norbert Manthey, Karl Nachtigall, Jens Opitz, and Peter Steinke. Solving periodic event scheduling problems with SAT. In He Jiang, Wei Ding, Moonis Ali, and Xindong Wu, editors, *Advanced Research in Applied Artificial Intelligence*, volume 7345 of *Lecture Notes in Computer Science*, pages 166–175. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-31086-7.

Peter Großmann, Jens Opitz, Reyk Weiß, and Michael Kümmling. On resolving infeasible periodic event networks. In *Conference on Advanced Systems in Public Transport*, 2015.

Leo Kroon and Leon Peeters. A variable trip time model for cyclic railway timetabling. *Transportation Science*, 37(2):198–212, 2003. doi: 10.1287/trsc.37.2.198.15247.

Michael Kümmling, Peter Großmann, Karl Nachtigall, Jens Opitz, and Reyk Weiß. A state-of-the-art realization of cyclic railway timetable computation. *Public Transport*, 7(3):281–293, 2015a. ISSN 1613-7159. doi: 10.1007/s12469-015-0108-5.

Michael Kümmling, Peter Großmann, Jens Opitz, Reyk Weiß, and Karl Nachtigall. Extraction of significant conflicts in periodic timetabling. In *Conference on Advanced Systems in Public Transport*, 2015b. URL `https://www.researchgate.net/publication/299409010_Extraction_of_Significant_Conflicts_in_Periodic_Timetabling`.

Christian Liebchen. Finding short integral cycle bases for cyclic timetabling. In Giuseppe Di Battista and Uri Zwick, editors, *Algorithms - ESA 2003*, volume 2832 of *Lecture Notes in Computer Science*, pages 715–726. Springer Berlin Heidelberg, 2003. ISBN 978-3-540-20064-2. doi: 10.1007/978-3-540-39658-1_64.

Christian Liebchen and Rolf H. Möhring. The modeling power of the periodic event scheduling problem: Railway timetables and beyond. In Frank Geraets, Leo Kroon, Anita Schoebel, Dorothea Wagner, and ChristosD. Zaroliagis, editors, *Algorithmic Methods for Railway Optimization*, volume 4359 of *Lecture Notes in Computer Science*, pages 3–40. Springer Berlin Heidelberg, 2007. ISBN 978-3-540-74245-6. doi: 10.1007/978-3-540-74247-0_1.

Christian Liebchen and Leon Peeters. Integral cycle bases for cyclic timetabling. *Discrete Optimization*, 6(1):98 – 109, 2009. ISSN 1572-5286. doi: http://dx.doi.org/10.1016/j.disopt.2008.09.003.

Christian Liebchen, Mark Proksch, and Frank H. Wagner. *Performance of Algorithms for Periodic Timetable Optimization*, pages 151–180. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. ISBN 978-3-540-73312-6. doi: 10.1007/978-3-540-73312-6_8. URL `https://doi.org/10.1007/978-3-540-73312-6_8`.

Karl Nachtigall. *Periodic network optimization and fixed interval timetables.* Habilitation thesis, Universität Hildesheim, 1999.

NS. Website nederlandse spoorwegen. `http://www.ns.nl`, 2017. Accessed: 2017-11-03.

Michiel Adriaan Odijk. A constraint generation algorithm for the construction of periodic railway timetables. *Transportation Research Part B: Methodological*, 30(6):455 – 464, 1996. ISSN 0191-2615. doi: http://dx.doi.org/10.1016/0191-2615(96)00005-7.

Michiel Adriaan Odijk, Ramon Lentink, and Adri Steenbeek. CADANS/Conflex: Functionele beschrijving van het onderdeel Conflex. Technical report, ORTEC Consultants BV, CWI, Gouda, Amsterdam, 2002.

Leon Peeters. *Cyclic Railway Timetable Optimization*. Phd thesis, Erasmus University Rotterdam, jun 2003.

Alexander Schrijver and Adri Steenbeek. Spoorwegdienstregelingontwikkeling. Technical report, CWI, Amsterdam, 1993. In Dutch.

Paolo Serafini and Walter Ukovich. A mathematical model for periodic scheduling problems. *SIAM Journal on Discrete Mathematics*, 2(4):550–581, 1989. doi: 10.1137/0402049.