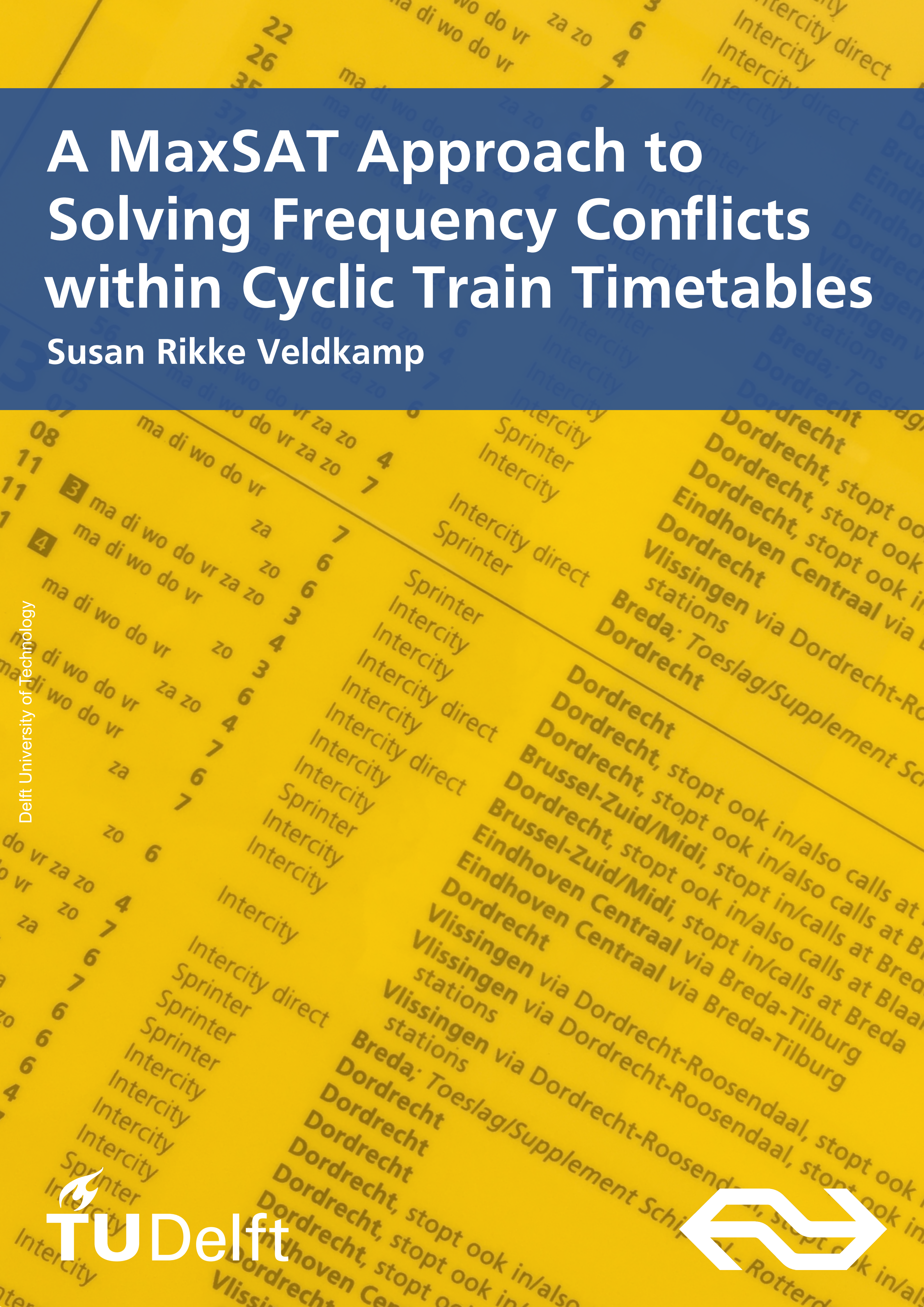


A MaxSAT Approach to Solving Frequency Conflicts within Cyclic Train Timetables

Susan Rikke Veldkamp

Delft University of Technology

TU Delft

A MaxSAT Approach to Solving Frequency Conflicts within Cyclic Train Timetables

by

Susan Rikke Veldkamp

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Friday November 25, 2022 at 10:00.

Student number:	4613775	
Project duration:	February 2022 – November 2022	
Thesis committee:	Dr. ir. L. J. J. Van Iersel,	TU Delft, supervisor
	Prof. Dr. Ir. C. Vuik,	TU Delft
	P.J. Fioole,	Nederlandse Spoorwegen, supervisor

Style: TU Delft Report Style, with modifications by Daan Zwaneveld

Preface

With this thesis I end my graduation project and my graduation internship at NS. I would like to express my thanks to some people that contributed to this thesis.
First of all, I want to thank my NS supervisor Pieter-Jan for guiding me and keeping me on track, giving good suggestions and reassuring me that my research was going somewhere. Of course I also want to thank my TU supervisor Leo for his help, meticulous proof-reading and interesting sparring sessions. Furthermore I want to thank Kees Vuik for being on my exam committee.

Next I would like to express gratitude to everybody working at PI for the good times, the warm welcome and all the pies. In particular, I want to thank Gabor, without whom I probably would have gotten lost in the Java libraries, and Maaike who was always ready to answer questions about her thesis or help me out with any Bitbucket troubles I had. Additionally, Danny, Marja and Camiel, thank you for your help, the not always serious but still fruitful discussions, and the games of fooseball! You made me enjoy being in the office every day.

Lastly I feel a great sense of gratitude toward my mother and father for supporting me throughout my studies. Papa, ik heb geprobeerd om zo min mogelijk komma's te gebruiken in dit verslag. Next to that I want to thank my sisters Dian and Lisette for always being examples I could follow.

I also want to thank my grandparents, with whom I lived for the first few months of my studies and where I was cared for by my lovely grandma.

Furthermore I would like to thank Wietse: thank you for the occasions you cooked for me after I came home from Utrecht.

Finally I want to thank my laptop for standing by me during what was probably 200+ hours of running code and 157+ times of refreshing Overleaf.

Susan Rikke Veldkamp
Delft, November 2022

Contents

Abstract

ILP-based solvers that intake a line planning, an infrastructure and a set of constraints and then output a cyclic train timetable have been researched extensively in the world of trains. However, these types of solvers are not able to automatically output a timetable when the problem gets overconstrained and no feasible solution exists.

It is possible to transform the timetabling problem into an equivalent Satisfiability problem that can be solved by SAT-solvers. In a recent thesis performed at NS, this has proven fruitful in cases where pre-chosen route choices prevent the model from finding a feasible solution. This is because the increased solving speed makes for a quicker discovery that no feasible solution is possible within the preset routes. It is then possible to iteratively call the solver, adding more route options in each iteration. By doing so, feasible timetables can be sought out within a reasonable time frame. Nonetheless, complications due to overconstraining still occur within these newer models, in particular when it comes to constraints that enforce train series to depart on set intervals, frequency constraints.

This thesis focuses on overcoming the problem of overconstraining due to frequency constraints and on consistently producing a timetable by using a maxSAT solver that can distinguish hard and soft constraints and assign them weights accordingly. Its aim is to minimize the total weight of violated soft constraints while maintaining all hard constraints, and this objective value can be compared to the total weight of violated clauses in solutions found by previous models. The resulting model does not always reach a better objective value after an hour of optimizing than the model it tries to improve. Nevertheless it is able to handle a set of conflicting constraints and always delivers a timetable that is as least as good as the one produced by the previous model. Additionally, it is capable of taking into account user preference for specific constraints through the weights.

Nomenclature

Abbreviations

Abbreviation	Definition	Page of first app.
BOT	Basic One-hourly Timetable	1
CADANS	Combinatorisch Algebraïsch Dienstregeling Algoritme NS	1
CNF	Conjunctive Normal Form	20
DONS	Designer Of Network Schedules	2
D	Drive	12
F	Frequency	16
FB	Frontaal Botsen (Frontal Collision)	15
II	In-In	13
ILP	Integer Linear Program	4
IU	In-Uit (In-Out)	14
maxSAT	The optimization problem Maximum Satisfiability	5
maxSAT_o	Model employing maxSAT solver and fixed orders within train series and within sets of trains on which shared frequency constraints are enforced	38
maxSAT_s	Model employing maxSAT solver and fixed orders within train series	33
maxSAT_x	Model employing maxSAT solver and no fixed orders of any trains	41
MIP	Mixed-Integer Program	6
MUS	Minimal unsatisfiable subformula	29
OPESP	Open Periodic Event Scheduling Problem	28
PEN	Periodic Event Network	9
PESP	Periodic Event Scheduling Problem	4
PF	Particular frequency	16
RvdK1	The model of Van der Knaap	39
RvdK2	The model of Van der Knaap without frequency constraints	39
S	Stopping	13
SAT	The feasibility problem Satisfiability	5
SF	Shared frequency	16
T	Transfer	17
TAM	Tool Aanpassing Materieelinzet	2
UI	Uit-In (Out-In)	14
UU	Uit-Uit (Out-Out)	13

Symbols

Symbol	Definition
$2^{\mathcal{I}_T}$	The power set of $\mathcal{I}_T$, i.e. the set of all subsets of $\mathcal{I}^T$
$\mathcal{A}$	The set of all constraints within a PESP
$a : \mathcal{V} \times \mathcal{V} \to 2^{\mathcal{I}_T}$	The function which assigns to each pair of nodes $(\varepsilon, \varepsilon') \in \mathcal{V} \times \mathcal{V}$ a set of intervals modulo T
$a(\varepsilon, \varepsilon')$	The set of intervals modulo T in which $d(\varepsilon) - d(\varepsilon')$ may lie
$d(\varepsilon_i^t)$	The departure time of train t from stage point i
$\mathcal{F}$	Set of trains on which a frequency constraint is enforced
$I : \Sigma_{SAT} \to \{\text{false}, \text{true}\}$	Interpretation that assigns each propositional variable in Σ_{SAT} true or false
$\mathcal{I}_T$	The set of intervals modulo T
$\mathcal{L}(\Sigma_{SAT})$	The smallest set that contains each propositional formula F such that for all $F \in \mathcal{L}(\Sigma_{SAT})$, F is unique.
m	margin allowed in Frequency constraints
$\mathcal{N} = (\mathcal{V}, T, a)$	Periodic Event Network
$q_{x,i}$	Boolean variable dictating that $x \leq i$.
$(r_t)_i$	Sequence of stage points followed by train t
s_t	Number of stage points on the route of train t
T	Period time
$\mathcal{V}$	Set of periodic events
y_i^t	Time it takes train t to drive from stage point i to stage point $i+1$
z_i^t	Parameter dictating whether or not train t has a stop between stage i and stage $i+1$
ε_i^t	The periodic event of train t leaving stage point i
$\zeta(\varepsilon, \varepsilon', c)$	The set of all infeasible line segments imposed by constraint c between the events ε and ε'.
$\xi_x(I)$	The function that extracts under an interpretation I from an order-encoded PESP-variable x the original value.
ϕ	Subset of F for F a SAT-formula in CNF
Σ_{SAT}	Alphabet of propositional logic
$\Psi_{ordered}^{\mathcal{N}}$	Propositional formula that order encodes all PESP constraints in PEN $\mathcal{N}$ into SAT-clauses
$\Omega_{ordered}^{\mathcal{N}}$	Propositional formula that order encodes all PESP variables in PEN $\mathcal{N}$ into SAT-clauses

1 Introduction

1.1. About Nederlandse Spoorwegen

The Netherlands has the busiest railway network in Europe: in a COVID free year the Dutch railway system transports about 1.3 million people daily [14]. The infrastructure is owned by the state and is managed by state company ProRail [1], but the management and operations of the largest part of the railway system is in hands of Nederlandse Spoorwegen (Dutch Railways, NS) [10]. To this end, NS employs approximately 19,000 people [15].

During peak hours, about 50% of all passenger trips managed by NS are between the four largest Dutch cities (Amsterdam, Rotterdam, The Hague, and Utrecht) [1]. If all these travelers made these trips by cars instead, this would cause a substantial congestion of the roads to these cities, making them hardly accessible anymore. Next to that, trains are more environmentally friendly than cars when it comes to green house emissions, even when the train uses fossil fuels [18]. Because of this the Dutch government advocates the growth of railway transport, but new infrastructure is expensive and takes up much space. Therefore NS tries to grow railway transport by optimizing the schedules for train operations on the already existing infrastructure [1].

1.2. The planning process at NS

NS follows several steps in making a train schedule [10]. First, a line plan is devised in which the direct connections and the frequencies of the trains get determined. NS takes the expected demand between every pair of stations and uses this to work out the train lines and their frequencies. While doing this NS tries to maximize direct connections and robustness and minimize the length of travel times. If a line plan is constructed for which no feasible timetable can be produced, the line plan has to be changed. Therefore this step and the next one depend heavily on each other.

Given a line plan, NS constructs a timetable that fills one hour, a so-called BOT (Basic Hourly Timetable). This timetable determines for each train and all stations the exact arrival and departure times such that constraints dictating for instance that no trains may crash into each other or that passengers should be able to catch transfers, are satisfied. To this end, NS uses the solver CADANS which creates the BOT. This solver either provides a feasible solution or an indication why the problem is infeasible. When a BOT is obtained, daily schedules are made by copy-pasting the hourly schedules, which are in turn copied to produce weekly schedules.

When the previous step is finished, rolling stock needs to be assigned to all trips in the timetable [9]. NS needs to assign trains to trips without using more rolling stock that they have at their disposal. While taking into account the composition of the trains, they balance passenger comfort and costs. To find a solution to the rolling stock scheduling problem, known as a rolling circulation, NS uses the MILP-based solver “TAM” which is based on a multi commodity flow representation [9]. If disruptions occur, NS needs to reschedule the rolling circulation, for which they use a neighborhood search heuristic.

When the trains are assigned to the trips, finally the crew can be assigned to the trains. All tasks are scheduled into feasible duties such that important constrictions like the maximal duty time and maximal number of nighttime duties are not exceeded and the prescribed division between so-called ‘sweet’ and ‘sour’ duties is met. The latter is based on the fact that some duties are more enjoyable than others, such as duties on lines that endure more agression or duties that are more monotonous than others. While taking all these constraints into account, NS tries to minimize the total number of duties needed to operate all trains by solving this as a set covering problem [10].

1.3. Timetable making process at NS

In this thesis we focus on the second step in the planning process: making timetables. BOTs are integral to this step. They need to be designed such that copy-pasting them over an entire day creates a cyclic timetable, which enables the customer to easily memorize train departure times and travel regularly.

The timetables of 2007 and 2017 were constructed with DONS, which builds the model and then employs the solvers CADANS and STATIONS. CADANS creates the BOT and STATIONS creates the basis platform assignment [10]. Since CADANS either provides a solution or an indication why the problem is infeasible, the user needs to change specifications when the problem is infeasible. Now, in 2022, as the Dutch train network has become busier, it has become harder to change the specifications to find a solution [20] by hand, even when supported by computer-aided design tools. Therefore it is desirable to produce a model that overcomes this problem.

1.4. Problem description

One of the cases in which CADANS will indicate that the timetabling problem is infeasible is in Example 1.4.0.1. Note that a set of trains is called a *train series* if all trains in it always travel the exact same route with the same stops along the route.

Example 1.4.0.1. Observe train series 1 and 2 in Figure 1.1. Train series 1 departs two times per hour while series 2 only leaves once every hour. On the tracks between station A and B, these two train series behave in exactly the same way, but after that train series 1 continues on to station C, while train series 2 travels to station D. Therefore, a passenger desiring to travel from station A to B can take a train 3 times an hour, while a passenger traveling to station C or station D can only get there twice or once an hour, respectively.

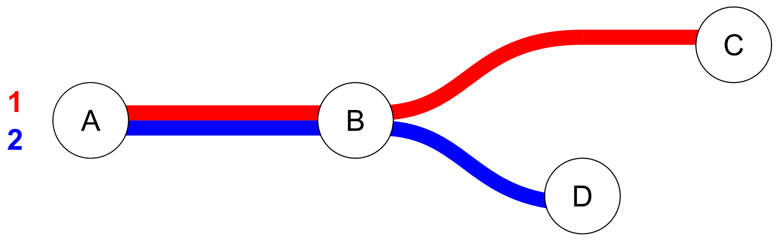
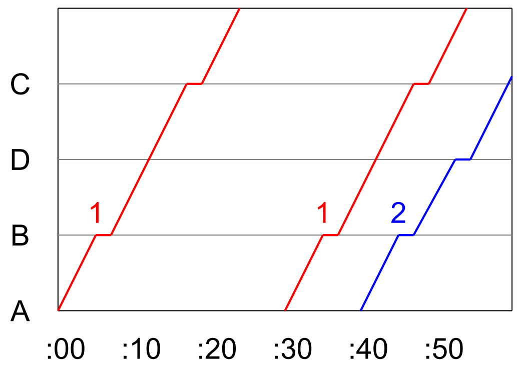


Figure 1.1: Train network of two different train series, 1 and 2, driving the same route on part of their journey. Both series 1 and 2 stop at A and B, but series 1 then travels to C while series 2 travels to D.

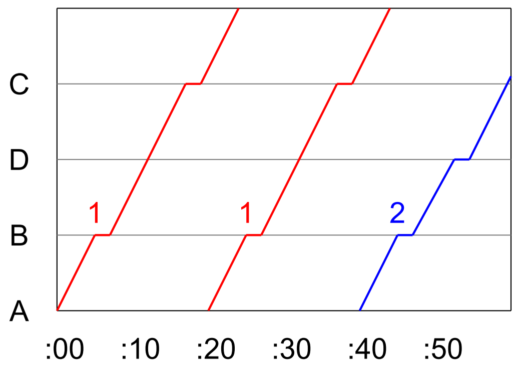
To ensure maximal comfort for their travelers, NS prefers trains that travel the same route and halt at the same stops to have a regular frequency. For instance, four trains are said to have a regular frequency if every fifteen minutes, one of them departs. Having trains depart regularly ensures that the travelers get divided evenly over the trains and that all travelers have to wait more or less the same amount of time.

Certainly all trains within one particular train series should have a regular frequency, as they call at all the same stations. However, when two different train series share the same route for part of their journey, it is desirable that they also have a regular frequency together on that part of the track. In our example, this would mean that the two train series together leave every 20 minutes as well. However, it is easy to see that this is not possible. If train series 1 leaves every 30 minutes, then train series 2 can not possibly leave both 20 minutes *after* a train of series 1 and 20 minutes *before* a train of series 1.

Similarly, if train series 1 and 2 together leave every 20 minutes, train series 1 cannot depart every 30 minutes. This is illustrated in the so-called time-distance diagrams in Figure 1.2. These diagrams show the movement of trains. The x-axis portrays the stations and the y-axis portrays the time. In Figure 1.2a, the focus lies on a regular distribution of train series 1, making it depart at :00 and :30. This causes train series 1 and 2 together to leave at :00, :30 and :40. In Figure 1.2b, where the train series together have an even 20-20-20 distribution, train series 1 leaves at :00 and :20, leaving a 40 minute gap between :20 and :00. In conclusion, it is just not possible to obey both constraints.



(a) Schedule in which train series 1 has a regular 30-30 frequency



(b) Schedule in which train series 1 and 2 have a regular 20-20-20 frequency together.

Figure 1.2: Two options for a time-distance diagram of train series 1 (red) and 2 (blue). In Figure 1.2a, train series 1 is neatly distributed. In Figure 1.2b, the combined train series is neatly distributed.

However, if we let train 1 from train series 1 stop at station B longer, then all travelers from station B are able to get to station C every half hour, and all travelers who desire to travel from A to B can do so every 20 minutes, such as in Figure 1.3. Note that travelers from station A still cannot travel to station C every half hour, as they can only board the train at station A at intervals of 20 and 40 minutes. Nevertheless, this train distribution seems to be a viable option. Unfortunately, trains are usually not allowed to stop at a station much longer than necessary. Stopping constraints enforce this. In fact, there are many more constraints that all need to be taken into account when divising a train timetable.

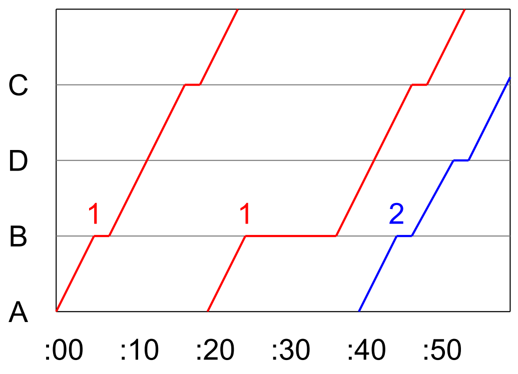


Figure 1.3: Schedule in which train series 1 leaves from station B regularly and train series 1 and 2 leave from station A regularly.

Enforcing a regular frequency on trains increases traveler comfort, but it is not as important as indispensable constraints such as requiring a safe distance between trains or imposing that trains pass the right stations on their trip in the first place. Therefore, since it is impossible to find a timetable that fulfills all these wishes, the problem tackled by this thesis is not finding a way to incorporate all constraints but to offer the next-best timetable by sacrificing less important constraints where need be.

NS employs a solver that takes in a train network and a line planning to ultimately find a timetable that honors safety constraints, trip-defining constraints and other requests imposed by the user, but the problem with these solvers is that (not surprisingly) they can only return such a timetable if it exists. If the timetable is requested to take into account the conflicting frequency constraints described above, the solver will return *infeasible*. Over the past two years, NS has ventured into a new way of solving these types of cyclic timetable finding problems, namely by transforming the feasibility problem into an equivalent feasibility problem called Satisfiability. Fast SAT-solvers make the process less time-consuming [8], and they are therefore better at the job than the old NS solver which relies on an ILP-solver. However, as SAT is a feasibility problem, SAT-solvers still do not return a timetable when the problem is over-constrained. Therefore the goal of this thesis is to answer the following question:

How can the SAT-model for the train timetabling problem be altered in such a way that it always outputs a timetable if one is possible within the strictly required constraints of the network?

We will now summarize literature relevant to the subject.

1.5. Literature review

The problem of finding a Basic One-Hourly Timetable (BOT) for trains is often described as a Periodic Event Scheduling Problem (PESP) , first introduced by Serafini and Ukovich [19] and outlined in Chapter 2.1. PESP is a feasibility problem that consists of trying to find a schedule that repeats each period and meets all pre-set constraints. These may be non-negotiable overhead constraints which enforce safe distances between trains on the same tracks, but also comfort constraints that, though not entirely essential, make timetables more appealing to travelers. Solving PESP therefore results in either finding a solution or rendering the problem infeasible. Hence, when no solution is found a new challenge arises: solving PESP such that all constraints are not obeyed definitely, but rather as closely as possible. As this NP-hard problem stands at the basis of all cyclic timetabling models, it has been researched extensively in the literature.

For instance, Polinder [16] describes how to produce a minimal relaxation of the constraints in order to obtain a feasible timetable through resolving smaller sets of conflicting constraints found by the NS PESP-solver CADANS one by one. Polinder's model relaxes the constraints within these conflicts minimally and then attempts to solve the entire problem again. This procedure gets repeated until a feasible timetable exists.

Furthermore, Lindner and Liebchen [12] devise a heuristic to find good solutions to PESP. These 'good' solutions are not actual solutions to PESP in that they do not meet all constraints PESP poses, but they meet a large part of them nonetheless. Lindner and Liebchen's heuristic computes several good solutions by solving PESP as an optimization problem that minimizes the deviations of the found solution from the lower bounds of the preset constraints. It then identifies variables that have similar values over all found solutions and tightens the bounds on these variables. Afterward, it solves the resulting optimization problem to obtain a better solution than the separate solutions and the final solution meets many of the original PESP constraints.

Polinder et al. [17] demonstrate how to solve the timetabling problem to optimality with regard to the total perceived travel time of all passengers without considering the infrastructure constraints. By viewing the perceived travel time instead of the absolute travel time the authors take into account that waiting is usually experienced as more bothersome by travelers than in-train time. After solving the problem the authors tweak the resulting timetable step by step such that the infrastructure constraints in the conflict are taken into account.

An efficient way of solving PESP is to transform the problem into an instance of Satisfiability (SAT) (see Chapter 2.3) through a polynomial reduction [6] (detailed in Chapter 2.4). The reason for this is that although both problems are NP-complete, SAT-solvers exist that are faster than state-of-the-art PESP-solvers [8], making it profitable to solve PESP as a SAT-problem.

Both Vollebergh [20] and Van der Knaap [11], two previous graduates at NS, developed timetable-producing models that use the SAT-formulation. Vollebergh's model overcomes overconstraining due to constraints that indicate a train is not allowed to drive on certain track options. This happens when the user presets a specific route that a train takes, including tracks. The model employs flexible track use by iteratively running the SAT-solver and adding extra track options for trains after the SAT-solver indicates in an iteration that no feasible solution is available. Van der Knaap uses a similar model but uses a SAT-solver that outputs information indicating where extra track options are likely useful. Therefore, instead of blindly adding extra track options on entire routes, her model only adds extra track options in certain places, saving computation time in further iterations by not unnecessarily increasing the number of options.

When a problem is overconstrained this is never due to any one constraint. Rather, it gets caused by a number of constraints that cannot be satisfied all at the same time, thereby conflicting with each other. From the studies of Polinder [16], Polinder et al. [17] and Lindner and Liebchen [12], it is clear that a good model avoids trying to satisfy conflicting constraints while aiming to make the timetable as comfortable for travelers as possible, for instance by enforcing train series to depart on regular intervals. The theses of Vollebergh and Van der Knaap suggest that continuing in the direction of the SAT-formulation of the timetabling problem may prove fruitful.

Vollebergh's and Van der Knaap's models both have one shortcoming: when no solution is available and all possible route options and track options have been exhausted, they are not able to forego the comfort constraints to produce a feasible timetable anyway. Therefore it is interesting to produce a model that incorporates certain comfort constraints only if it results in a feasible timetable, using Maximum-Satisfiability or maxSAT. This is the optimization version of Satisfiability. It gets introduced in its full extent in Chapter 2.3, along with Weighted Partial maxSAT. The nature of (Weighted Partial) maxSAT allows for solutions that do not necessarily meet all constraints.

Most maxSAT solvers employ repeated calls to a SAT-solver [2], hence they may inherit some of that speed. Grossmann [7] optimizes PESPs extended by decisional transport flow networks for several instances of the German train network by transforming the instances into maxSAT. The maxSAT solver outperforms the MIP solver on most instances. Since it does so well on decision flow PESPs, this indicates that a maxSAT solver may perform well on PESPs with fewer decision variables as well.

Lastly, although it seems evident that including more comfort constraints would increase passenger comfort, it is important to check how significant this improvement in passenger comfort is. In their research, Polinder et al. [17] calculate the total perceived travel time as a function of the timetable and used this to measure the amount of comfort within any timetable. To compute this, it is necessary to valuate the ‘costs’ of waiting time and transfers compared to regular traveling time. What these costs should be has been investigated by Bakker, P. et al. [3]. Polinder et al. make several assumptions on traveler’s behaviour, such as that all passengers take the shortest possible routes. They further assume that passengers arrive randomly and at a constant rate at their departure station. According to Bakker et al. [3], this assumption is often made in Dutch timetabling models. This is quite a realistic assumption if we view the random arrival as a wish to depart at a certain time and in the case that travelers arrive from a different train to transfer to another. This makes it possible to compute the total waiting time as a simple function of this arrival rate, as explained by Daganzo [5].

1.6. Outline

This thesis proposes an addition to the model of Van der Knaap that ensures that there is always a timetable produced that adheres to all strictly required constraints. It starts with a description of the PESP-SAT model employed by Van der Knaap in Chapter 2, before explaining how maxSAT is incorporated in Van der Knaap’s model in Chapter 3. After that, an overview of the results will be given in Chapter 4 prior to a discussion and a conclusion in Chapter 5.

2
The PESP-SAT timetabling model and Van der Knaap's application of it

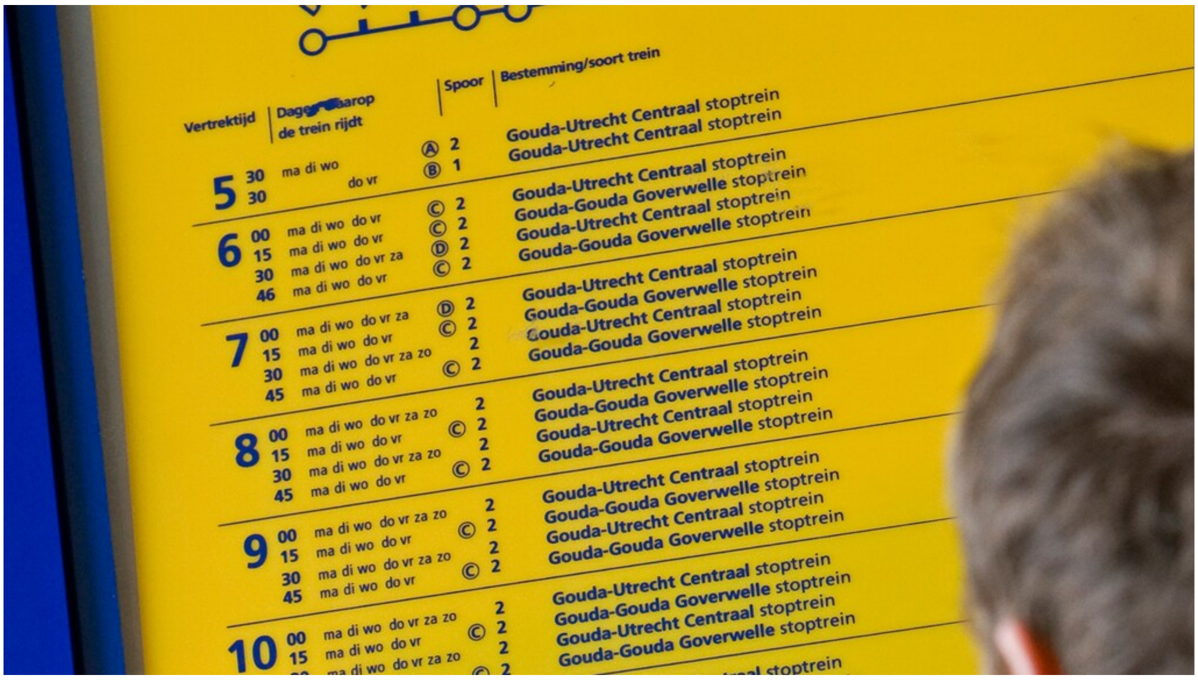
The model used in this research was adapted from the model used by Van der Knaap [11], which was in turn based on Vollebergh's model [20]. This model finds its basis in the Periodic Event Scheduling Problem (PESP) which was first introduced by Serafini and Ukovich [19], and is often used to describe the problem of finding a cyclic timetable. Below we outline the basic principles of a general Periodic Event Scheduling Problem.

2.1. The Periodic Event Scheduling Problem

In the *Periodic Event Scheduling Problem* we have a set of events in which each event repeats every period time T. In the Netherlands train timetables have period 60 and the periodic events are trains departing from stations, such that the same train departures repeat each hour like in the timetable in Figure 2.1. Recall that a set of trains that travel the exact same route and call at the exact same stations is called a *train series*, and it may depart multiple times an hour. In this case, these departures are all different periodic events that repeat each hour.

More generally, Serafini and Ukovich [19] define a periodic event as follows.

Definition 2.1.0.1 (Periodic event). A *periodic event* ε is a set of events e_p indexed by $p \in \mathbb{Z}$, with variable departure times $d(e_p) \in \mathbb{R}$, such that for each p, $d(e_p) - d(e_{p-1}) = T$. T is referred to as the *period* and e_p as the *pth occurrence* of the periodic event ε. We also define $d(\varepsilon) := d(e_p) \mod T$.



Vertrektijd		Dagen waarop de trein rijdt	Spoor	Bestemming/soort trein
5	30	ma di wo	Ⓐ 2	Gouda-Utrecht Centraal stoptrein
	30	do vr	Ⓑ 1	Gouda-Utrecht Centraal stoptrein
6	00	ma di wo do vr	Ⓒ 2	Gouda-Utrecht Centraal stoptrein
	15	ma di wo do vr	Ⓒ 2	Gouda-Gouda Goverwelle stoptrein
	30	ma di wo do vr za	Ⓓ 2	Gouda-Utrecht Centraal stoptrein
	46	ma di wo do vr	Ⓒ 2	Gouda-Gouda Goverwelle stoptrein
7	00	ma di wo do vr za	Ⓓ 2	Gouda-Utrecht Centraal stoptrein
	15	ma di wo do vr	Ⓒ 2	Gouda-Gouda Goverwelle stoptrein
	30	ma di wo do vr za zo	2	Gouda-Utrecht Centraal stoptrein
	45	ma di wo do vr	Ⓒ 2	Gouda-Gouda Goverwelle stoptrein
8	00	ma di wo do vr za zo	2	Gouda-Utrecht Centraal stoptrein
	15	ma di wo do vr	Ⓒ 2	Gouda-Gouda Goverwelle stoptrein
	30	ma di wo do vr za zo	2	Gouda-Utrecht Centraal stoptrein
	45	ma di wo do vr	Ⓒ 2	Gouda-Gouda Goverwelle stoptrein
9	00	ma di wo do vr za zo	2	Gouda-Utrecht Centraal stoptrein
	15	ma di wo do vr	Ⓒ 2	Gouda-Gouda Goverwelle stoptrein
	30	ma di wo do vr za zo	2	Gouda-Utrecht Centraal stoptrein
	45	ma di wo do vr	Ⓒ 2	Gouda-Gouda Goverwelle stoptrein
10	00	ma di wo do vr za zo	2	Gouda-Utrecht Centraal stoptrein
	15	ma di wo do vr	Ⓒ 2	Gouda-Gouda Goverwelle stoptrein
		ma di wo do vr za zo	Ⓒ 2	Gouda-Utrecht Centraal stoptrein
				Gouda-Gouda Goverwelle stoptrein
				al stoptrein

Figure 2.1: NS train timetable displayed on one of the so-called "gele borden" (yellow displays). A train that departs at 6:15 will also depart at 7:15, 8:15 and so on. Source: RTL Nieuws/ANP [13]

A periodic timetable such as above is an assignment of time values within the interval $[0, T)$ to all the events, such that the (periodic) time differences between certain pairs of events meet a feasibility interval [12]. These feasibility intervals stem from constraints imposed on the PESP, such as the constraint in the following example.

Example 2.1.0.1. Trains t and t' both leave station Delft, platform 1 once each hour. Let ε_t denote the periodic event that train t leaves, and $\varepsilon_{t'}$ the periodic event that train t' leaves. For safety reasons, the trains' departures need to be three minutes or more apart. If train t leaves before train t' this means that $d(\varepsilon_{t'}) - d(\varepsilon_t) \geq 3$. However, because the trains leave each hour, train t also departs *after* train t' in the following hour, and hence train t' must also be three minutes earlier than train t, such that $d(\varepsilon_t) - d(\varepsilon_{t'}) \geq 3$. For this constraint, $d(\varepsilon_{t'}) - d(\varepsilon_t) \in [3, 57]$ is the corresponding feasibility interval.

More generally, since in PESPs the order between two periodic events is unknown beforehand, a constraint upon two periodic events gets defined as below for l, u respective lower and upper bounds and T the period.

Definition 2.1.0.2 (Constraint)**.** Let $l, u \in \mathbb{R} : l \leq u, T \in \mathbb{Z}^+$ and let $d(\varepsilon)$ indicate the departure time of a periodic event ε. A *constraint* a is a relationship involving an ordered pair of periodic events $(\varepsilon, \varepsilon')$ requiring that

$$d(\varepsilon) - d(\varepsilon') \in [l, u]_T = \bigcup_{k \in \mathbb{Z}} [l + kT, u + kT]$$

Here, $[l, u]_T$ is called an *interval modulo* T.

Let $\mathcal{I}_T$ be the set of intervals modulo T. An instance of PESP is given in the definition of a Periodic Event Network.

Definition 2.1.0.3 (Periodic Event Network)**.** A *periodic event network* (PEN) is a triple $\mathcal{N} = (\mathcal{V}, T, a)$ with $\mathcal{V}$ the set of periodic events, $T \in \mathbb{N}$ the period and $a : \mathcal{V} \times \mathcal{V} \to 2^{\mathcal{I}_T}$ a function which assigns to each pair of periodic events $(\varepsilon, \varepsilon') \in \mathcal{V} \times \mathcal{V}$ a set of intervals modulo T. Then for each $\varepsilon, \varepsilon' \in \mathcal{V}$, $a(\varepsilon, \varepsilon')$ is a set of constraints.

To create a clear overview, a PEN can be displayed as a directed multigraph where the nodes are the periodic events $\mathcal{V}$ and for all $c \in a(\varepsilon, \varepsilon')$ for all $(\varepsilon, \varepsilon') \in \mathcal{V} \times \mathcal{V}$ a directed edge exists from ε to ε', labelled by the interval c, such as in Figure 2.5a. To solve a PESP is to find a valid schedule for the given problem.

Definition 2.1.0.4 (Schedule)**.** A *schedule* is a function $d : \mathcal{V} \to \{0, \ldots, T-1\}$. A schedule d is *valid with respect to* a if and only if

$$d(\varepsilon') - d(\varepsilon) \in \bigcap_{c \in a(\varepsilon, \varepsilon')} c.$$

Definition 2.1.0.5 (Periodic Event Scheduling Problem)**.** Given a PEN, find a *schedule* $\tau(\varepsilon)$ for each $\varepsilon \in N$ that satisfies all constraints.

2.1.1. Two properties of PESP constraints

As we saw before, PESP gets governed by constraints which it needs to obey. A constraint function $a(\varepsilon, \varepsilon')$ assigns a pair of events ε and ε' some interval modulo T. One property of PESP constraints is that any constraint defined on an ordered pair $(\varepsilon, \varepsilon')$ is easily transformed into a constraint on the ordered pair $(\varepsilon', \varepsilon)$ in the following manner:

$$\begin{aligned} d(\varepsilon', \varepsilon) \in [l, u]_T &\iff d(\varepsilon, \varepsilon') \in [-u, -l]_T \\ &\iff d(\varepsilon, \varepsilon') \in [-u + T, -l + T]_T. \end{aligned} \tag{2.1}$$

Furthermore, within PESP it is possible to assign a pair of periodic events a constraint that is made up of disjoint intervals such as below:

$$d(\varepsilon) - d(\varepsilon') \in [l_1, u_1]_T \cup [l_2, u_2]_T. \tag{2.2}$$

Because $[l_1, u_1]_T \cup [l_2, u_2]_T$ is equivalent to $[l_1, u_2]_T \cap [l_2, u_1 + T]_T$, as can be seen in Figure 2.2, constraint (2.2) can be turned into two separate constraints as follows:

$$\begin{gathered} d(\varepsilon) - d(\varepsilon') \in [l_1, u_1]_T \cap [l_2, u_2 + T]_T \\ \iff \\ d(\varepsilon) - d(\varepsilon') \in [l_1, u_2]_T \;\wedge\; d(\varepsilon) - d(\varepsilon') \in [l_2, u_1 + T]_T. \end{gathered}$$

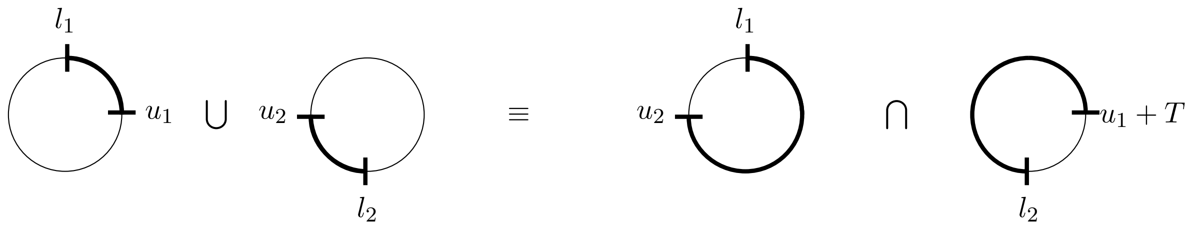


Figure 2.2: Illustration of how a union of disjoint intervals can be written as an equivalent intersection of intervals

This idea is generalized in Lemma 2.1.1.1.

Lemma 2.1.1.1. Suppose that for some constraint $a(\varepsilon, \varepsilon') \in \mathcal{A}$ we want to impose the constraint

$$d(\varepsilon') - d(\varepsilon) \in [l_1, u_1]_T \cup [l_2, u_2]_T \cup \ldots \cup [l_k, u_k]_T$$

where the k intervals are disjoint and ordered such that

$$0 \leq l_1 \leq u_1 < l_2 \leq u_2 < \ldots < l_k \leq u_k < l_1 + T.$$

Then this union of k intervals is equal to the intersection of k intervals modulo T given by the constraints

$$\begin{aligned} [l_1, u_k]_T &\in a(\varepsilon, \varepsilon'), \\ [l_2, u_1 + T] + T &\in a(\varepsilon, \varepsilon'), \\ &\vdots \\ [l_k, u_{k-1} + T]_T &\in a(\varepsilon, \varepsilon'). \end{aligned}$$

The proof of this lemma can be found as an Appendix in Vollebergh's thesis [20].

We are now ready to formulate PESP for our particular goal: finding a train timetable.

2.2. PESP formulation for train timetables

To formulate PESP for train timetables, we first need to go over some administration of our train network and line planning. We therefore introduce the concept of stage points. The infrastructure of our network consists of tracks and stage points. Stage points are points that naturally divide the routes of trains into parts that we will refer to as *stages*. A stage point is not only defined by its stage point name (for instance, a station), but also by which track the stage point is on. Therefore a stage point $i = (i_1, i_2) \in \mathbb{N} \times \mathbb{N}$, where i_1 denotes the stage point name and i_2 denotes the track. Stage points can be on stations, but there are also artificial stage points put in before switches to ensure that there is always only one option from one stage point to another.

2.2.1. Variables

We will now define the variables. For N the set of trains in our line planning, each train $t \in N$ has a route r_t that consists of a sequence of stage points $(r_t)_i$. This route is particular to that train, and therefore stage points $(r_t)_i$ can only lie on the route that is set for that train, within which the route points and particular tracks are pre-defined. For instance, Figure 2.3 shows two train routes r_t (orange) and $r_{t'}$ (blue). Because the stage points are predefined, we know which stage points names and tracks train t travels on. Therefore train t travels from $(r_t)_1 = (A, 1)$ to $(r_t)_2 = (B, 1)$ to $(r_t)_3 = (C, 1)$.

The total number of stage points that train t passes on its route is $s_t \in \mathbb{N}$, and that includes the beginning point and end point.

Furthermore, $y_i^t \in \mathbb{N}$ is the technical driving time that it takes to drive from stage point i to the next stage point $i+1$. It is the time between train t departing stage point i and train arriving at stage point $i+1$, so no waiting time is included. The driving time is dependent on the train t that drives the stage, as different train types have different speeds. Lastly, each stage has a parameter $z_i^t \in \{0, 1\}$ such that

$$z_i^t = \begin{cases} 1 & \text{if train } t \text{ has a stop between stage } i \text{ and stage } i+1; \\ 0 & \text{otherwise.} \end{cases}$$

This parameter is in place such that the necessary constraints can be assigned to the right train departures.
Now we can define a periodic event for each train and each stage point on its route:

$$\varepsilon_i^t := \text{the event that train } t \text{ departs from stage point } (r_t)_i.$$

The set of periodic events then becomes

$$\mathcal{V} = \{\varepsilon_i^t : t = 1, \ldots, N; i = 1, \ldots, s_{t-1}\}.$$

We illustrate the above variables in Example 2.2.1.1, accompanied by Figure 2.3.

Example 2.2.1.1. Consider the line planning of two trains t and t'. Train t travels from station A to station C without stopping at station B, while a second train t' leaves station A and does stop at station B before arriving at station C. Train t arrives at the left-hand side of the platform of station C, while train t' arrives at the right-hand side. This is represented in Figure 2.3. We go over all the parameters that arise from this scenario to make the reader familiar with their meanings.

- The numbers of stage points are $s_t = s_{t'} = 3$, because both trains pass 3 stations.
- The routes are $r_t = \{(A, 1), (B, 1), (C, 1)\}$, but $r_{t'} = \{(A, 1), (B, 1), (C, 2)\}$ because the trains use different tracks on their last stage.
- The technical driving times are $y_A^t = 3; y_B^t = 5; y_A^{t'} = 4; y_B^{t'} = 6$. Note that since train t' has to call at station B, it takes a longer time to travel from station B to station C, and longer to travel form station A to B than train t, even though they travel on the same tracks.
- The stopping parameters are $z_A^{t'} = 1$ and $z_B^{t'} = 1$ as train t' stops on all stations, but $z_A^t = 1$ and $z_B^t = 0$ as train t does not halt on station B.
- The periodic events that define the trains are now: $\mathcal{V} = \{\varepsilon_A^t, \varepsilon_B^t, \varepsilon_A^{t'}, \varepsilon_B^{t'}\}$.

Note that since it is predefined that train t' uses the first option of the two tracks between stage points B and C, $\varepsilon_B^{t'}$ indicates the travel of train t' from stage point B on track 2 to the bottom stage point C.

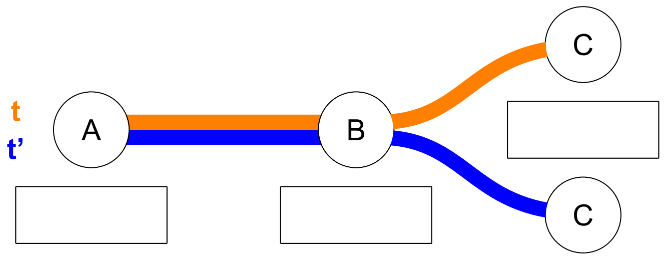


Figure 2.3: Example of a small railway network with stage points A,B, and C, where the rectangles signify platforms and the colored lines the routes that the trains take. Here train t (in orange) and train t' (in blue) both follow stage points A, B and C.

Now that all the necessary parameters are explained, we can formulate the constraints necessary for our PESP model.

2.2.2. Constraints

Remember that a constraint can be defined between any ordered pair of periodic events $(\varepsilon_i^t, \varepsilon_{i'}^{t'})$ by assigning the pair an interval modulo T. The set of intervals modulo T that signify the constraints between an ordered pair of periodic events is denoted as $a(\varepsilon_i^t, \varepsilon_{i'}^{t'})$, and when an interval modulo T $[l, u]_{60} : l \leq u \in \mathbb{R}$ is added to the set $a(\varepsilon_i^t, \varepsilon_{i'}^{t'})$, this indicates that a constraint of the following form is set on the two periodic events:

$$d(\varepsilon_i^t) - d(\varepsilon_{i'}^{t'}) \in [l, u]_{60},$$

where $d(\varepsilon_i^t)$ denotes the departure time of the periodic event that train t leaves from stage point i.

Some constraints below deal with the event of a train *arriving*. Although our model is only equipped with periodic events that signify *departures* of trains, we can easily enforce constrictions on arrival times as well. Since all technical travel times are fixed for a certain train and a certain stage, a train t's arrival time on stage point $i+1$ is equal to the time train t departs from the previous stage point i plus the technical travel time it takes to get to stage point $i+1$:

$$d(\varepsilon_i^t) + y_i^t =: \text{the arrival time of train } t \text{ at stage point } i+1.$$

In the following sections, the types of constraints are described and then formulated in logical statements. There are three main types of constraints.

Trip-defining constraints

Because a train journey consists of several consecutive periodic events that state a train leaves a certain stage point, periodic events belonging to one train must succeed each other without too much slack time. Furthermore, when trains are set to call at a station, they should halt at that particular station for a certain amount of time. These constraints are only enforced on periodic events belonging to the same train, and since they define the trip of a train they are therefore integral to the model. There are two types of trip-defining constraints:

- **D-constraints.** When the train does not stop in between two stages, the difference between the departure times at route points $(r_t)_i$ and $(r_t)_{i+1}$ should be the technical travel time plus some room for slack. The train must linger for a minimal slack time `minSlackTime` $\in \mathbb{N}$ and it is allowed to linger for a maximal slack time $\texttt{maxSlackTime} \in \mathbb{N}$. This type of constraint is called a *D-constraint* (Drive constraint). Stated in our variables, we obtain:

For all $t = 1, \ldots, N$ and $i = 1, \ldots, s_t - 2$

$$z_i^t = 0 \Rightarrow [y_i^t + \texttt{minSlackTime}, y_i^t + \texttt{maxSlackTime}]_{60} \in a(\varepsilon_i^t, \varepsilon_{i+1}^t).$$

- **S-constraints.** When the train is destined to stop between two stages, the difference between the departure times has to be at least the technical travel time plus a minimal amount of time $\texttt{minStopTime} \in \mathbb{N}$ so people can get on and off the train. It may at most be the technical travel time plus a maximal stopping time $\texttt{maxStopTime} \in \mathbb{N}$ so that it does not create delays. This is called an *S-constraint* (Stopping constraint). In other words,

  For all $t = 1, \ldots, N$ and $i = 1, \ldots, s_t - 2$,

$$z_i^t = 1 \Rightarrow [y_i^t + \texttt{minStopTime}, y_i^t + \texttt{maxStopTime}]_{60} \in a(\varepsilon_i^t, \varepsilon_{i+1}^t).$$

Safety constraints

Next to trip-defining constraints, there is another type of constraints that is critical to the model: safety constraints. These are in place such that trains that travel (in part) the same routes are always a safe distance apart and do not collide anywhere. For one, all trains need to have a safety headway $\texttt{minHeadwayTime}$ which makes trains on the same tracks and same stage point always have some time between them. Constraints come in several types for every manner that a pair of different trains could obstruct each other:

- **UU-constraints.** When two different trains both depart from the same stage point and same track, the difference between their departure times should be a safe headway duration $\texttt{minHeadwayTime} \in \mathbb{N}_{>0}$. This is enforced in a *UU-constraint* (Out-Out constraint, *Uit-Uit* in Dutch).

  And so, for all $t, t' = 1, \ldots, N, t \neq t'$ and $i = 1, \ldots, s_t - 1$ and $i' = 1, \ldots s_{t'} - 1$, if

$$(r_t)_i = (r_{t'})_{i'},$$

$$\begin{aligned}
&\Big(d(\varepsilon_{i'}^{t'}) - d(\varepsilon_i^t) \in [\texttt{minHeadwayTime}, 59]_{60}\Big) \wedge \Big(d(\varepsilon_i^t) - (d(\varepsilon_{i'}^{t'}) \in [\texttt{minHeadwayTime}, 59]_{60}\Big) \\
\Leftrightarrow\ & d(\varepsilon_{i'}^{t'}) - d(\varepsilon_i^t) \in [\texttt{minHeadwayTime}, 59]_{60} \cap [-59, -\texttt{minHeadwayTime}]_{60} \\
\Leftrightarrow\ & d(\varepsilon_{i'}^{t'}) - d(\varepsilon_i^t) \in [\texttt{minHeadwayTime}, 60 - \texttt{minHeadwayTime}]_{60},
\end{aligned}$$

  where the first equivalence follows from Property 2.1 and the second equivalence from the fact that $[-59, -\texttt{minHeadwayTime}]_{60} \equiv [1, 60 - \texttt{minHeadwayTime}]_{60}$. Therefore we get that

$$(r_t)_i = (r_{t'})_{i'} \Rightarrow [\texttt{minHeadwayTime}, 60 - \texttt{minHeadwayTime}]_{60} \in a(\varepsilon_i^t, \varepsilon_{i'}^{t'}).$$

- **II-constraints.** These constraints are similar to the UU-constraints. When two different trains both arrive at the same stage point on the same track, the difference between their arrival times should be at least $\texttt{minHeadwayTime}$. Therefore the difference in time between their departure of the previous stage point on their route plus the technical travel time should be at least $\texttt{minHeadwayTime}$.

For all $t, t' = 1, \dots, N, t \neq t'$ and $i = 1, \dots, s_t - 1$ and $i' = 1, \dots s_{t'} - 1$, if

$$(r_t)_{i+1} = (r_{t'})_{i'+1},$$

then

$$\begin{aligned} &\left(d(\varepsilon_{i'}^{t'}) + y_{i'}^{t'}\right) - \left(d(\varepsilon_i^t) + y_i^t\right) \in [\texttt{minHeadwayTime}, 60 - \texttt{minHeadwayTime}]_{60} \\ \Leftrightarrow \quad & d(\varepsilon_{i'}^{t'}) - d(\varepsilon_i^t) \in [\texttt{minHeadwayTime} + y_i^t - y_{i'}^{t'}, 60 - \texttt{minHeadwayTime} + y_i^t - y_{i'}^{t'}]_{60}. \end{aligned}$$

Consequently,

$$(r_t)_{i+1} = (r_{t'})_{i'+1} \Rightarrow [\texttt{minHeadwayTime}+y_i^t-y_{i'}^{t'}, 60-\texttt{minHeadwayTime}+y_i^t-y_{i'}^{t'}]_{60} \in a(\varepsilon_i^t, \varepsilon_{i'}^{t'}).$$

- **UI-constraints.** When one train departs from a certain stage point on a certain track and another arrives on this same stage point and track, these events need to take place at least `minHeadwayTime` minutes apart from eachother. Put differently, the other train's departure from the previous stage point plus travel time should be at least three minutes apart from the one train's departure. This is implemented in a *UI-constraint*, Dutch for *Uit-In* (Out-In).

  Hence, for all $t, t' = 1, \dots, N, t \neq t', i = 1, \dots, s_t - 1$ and $i' = 1$ if

  $$(r_t)_{i+1} = (r_{t'})_{i'},$$

  $$\begin{aligned} &\left(d(\varepsilon_{i'}^{t'}) + y_{i'}^{t'}\right) - d(\varepsilon_i^t) \in [\texttt{minHeadwayTime}, 59]_{60} \\ \Leftrightarrow \quad & d(\varepsilon_{i'}^{t'}) - d(\varepsilon_i^t) \in [\texttt{minHeadwayTime} - y_{i'}^{t'}, 59 - y_{i'}^{t'}]_{60}. \end{aligned}$$

  Therefore we have

  $$(r_t)_i = (r_t')_{i'+1} \Rightarrow [\texttt{minHeadwayTime} - y_{i'}^{t'}, 59 - y_{i'}^{t'}]_{60} \in a(\varepsilon_{i'}^{t'}, \varepsilon_i^t).$$

- **IU-constraints.** Similarly, when one train arrives at a stage point and track and another train departs from this one stage point and track, we need to set a minimal time-difference of `minHeadwayTime`. In other words, there needs to be `minHeadwayTime` between the departure of the first train from their current stage point plus the technical travel time it takes to get to the next stage point and the departure time of the second train. We enforce this in an *IU-constraint*, (*In-Uit* in Dutch), an In-Out constraint.

  For all $t, t' = 1, \dots, N, t \neq t', i = 1, \dots, s_t - 1$ and $i' = 1, \dots, s_{t'}$, if

  $$(r_t)_{i+1} = (r_{t'})_{i'},$$

  $$\begin{aligned} &\left(d(\varepsilon_i^t) + y_i^t\right) - d(\varepsilon_{i'}^{t'}) \in [\texttt{minHeadwayTime}, 59]_{60}, \\ \Leftrightarrow \quad & d(\varepsilon_i^t) - d(\varepsilon_{i'}^{t'}) \in [\texttt{minHeadwayTime} - y_i^t, 59 - y_i^t]_{60}, \\ \Leftrightarrow \quad & d(\varepsilon_{i'}^{t'}) - d(\varepsilon_i^t) \in [1 + y_i^t, 60 - \texttt{minHeadwayTime} + y_i^t]_{60} \quad \text{by Prop. 2.1} \end{aligned}$$

  such that

  $$((r_t)_{i+1} = (r_{t'})_{i'} \Rightarrow [1 + y_i^t, 60 - \texttt{minHeadwayTime} + y_i^t]_{60} \in a(\varepsilon_{i'}^{t'}, \varepsilon_i^t).$$

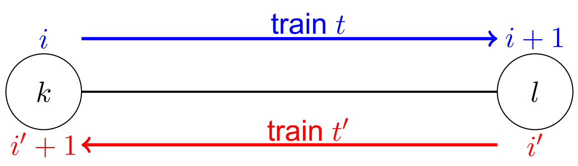


Figure 2.4: Visualisation of the event that Equation (2.7) happens. The red arrow illustrates train t' and the blue arrow train t, who have as their next stage point the other train's current stage point.

- **FB-constraints.** When two trains t and t' both travel on the same stage in opposite directions and they are set to crash into each other, this needs to be prevented through an *FB-constraint* (Frontal Collision, *Frontaal Botsen* in Dutch). This situation is depicted in Figure 2.4. We have given the stage points universal names k and l for clarity.

  An FB-constraint makes sure of two things: (1) that the departure time of train t from stage point k is later than the arrival time of train t' at stage point k, therefore train t' can continue its journey before train t arrives at stage point k. And (2) that the departure time of train t' from stage point l is later than the arrival time of train t at stage point l, because then train t can already have left stage point l before train t' sets out to depart that stage point.

  We have that "train t must *depart* from stage point k later than train t' will *arrive* at stage point k". In our variables, this becomes:

$$d(\varepsilon_k^t) > d(\varepsilon_l^{t'}) + y_l^{t'} \Leftrightarrow d(\varepsilon_k^t) - d(\varepsilon_l^{t'}) > y_l^{t'}. \tag{2.3}$$

  Changing the universal names back to the accurate stage points on the respective routes of the trains, the right hand side of Equation 2.3 is equivalent to the following:

$$d(\varepsilon_i^t) - d(\varepsilon_{i'}^{t'}) > y_{i'}^{t'} \Leftrightarrow d(\varepsilon_i^t) - d(\varepsilon_{i'}^{t'}) \in [y_{i'}^{t'}, 60]_{60}, \tag{2.4}$$

  where the equivalence in Equation 2.4 is due to the fact that the difference between two departures occurring hourly can at most be 60 minutes apart.

  The second statement reads: "Train t' must *depart* from stage point l later than train t will *arrive* at stage point l". We can capture this in our variables in the same way as described above.

$$d(\varepsilon_l^{t'}) > d(\varepsilon_k^t) + y_k^t \Leftrightarrow d(\varepsilon_{i'}^{t'}) - d(\varepsilon_i^t) \in [y_i^t, 60]_{60} \tag{2.5}$$

$$\text{By property 2.1, } \Leftrightarrow d(\varepsilon_i^t) - d(\varepsilon_{i'}^{t'}) \in [0, 60 - y_i^t]_{60}, \tag{2.6}$$

  Collecting constraints 2.4 and 2.6 into one gives us the following:

$$\begin{aligned} d(\varepsilon_i^t) - d(\varepsilon_{i'}^{t'}) &\in [y_{i'}^{t'}, 60]_{60} \cap [0, 60 - y_i^t]_{60} \\ &\in [y_{i'}^{t'}, 60 - y_i^t]_{60}. \end{aligned}$$

  We can now summarize these constraints into logical statements:

  For all $t, t' = 1, \ldots, N, t \neq t', i = 1, \ldots, s_t - 1$ and $i' = 1, \ldots, s_{t'} - 1$, if

$$(r_t)_i = (r_{t'})_{i'+1} \wedge (r_t)_{i+1} = (r_{t'})_{i'}, \text{ such as in Figure 2.4} \tag{2.7}$$

then

$$[y_i^t, 60 - y_{i'}^{t'}]_{60} \in a(\varepsilon_{i'}^{t'}, \varepsilon_i^t).$$

Comfort constraints

Next to necessary restrictions such as trip-defining constraints and safety constraints, there is a third type of constraints which is devised to increase traveler comfort. Rather than modelling absolute necessities in the train timetable, comfort constraints ensure that passengers can travel regularly or easily catch transfers. These are exactly the two types of comfort constraints.

- **F-constraints.** When a set of trains $\mathcal{F}$ share part of their route or their entire route, we can enforce that they are largely equally distributed across the period, say with a margin $m \in \mathbb{Z}$. Remember that a train series is a set of trains that travel the exact same route and call at the exact same stage points. Therefore, an example of enforcing Frequency constraints would be asking a train series with a frequency of 2 to leave approximately every 30 minutes. This wish is imposed by an *F-constraint*, a Frequency constraint.

  Let q be the first stage point that the trains in $\mathcal{F}$ have in common. Then for all $t, t' \in \mathcal{F}, t \neq t', i = 1, \ldots, s_t - 1, i' = 1, \ldots, s_{t'} - 1$, if

$$(r_t)_i = (r_{t'})_{i'} \;\; \forall i, i' = q, \ldots, p,$$

$$d(\varepsilon_q^{t'}) - d(\varepsilon_q^t) \in \bigcup_{k=1}^{n-1} \left[\frac{k}{n}60 - m, \frac{k}{n}60 + m\right]_{60} \text{ for } n = |\mathcal{F}|.$$

  Note that we only enforce these intervals on the departure of the trains t and t' from stage point q, as their driving times are invariable and therefore a train departing in a regular frequency at a certain stage point will depart in a regular frequency on all other stage points.

  The above union of intervals consists of $n - 1$ disjoint intervals, if and only if

$$\frac{k}{n}60 + m < \frac{k+1}{n}60 - m \Leftrightarrow 2m < \frac{60}{n}.$$

  We use Lemma 2.1.1.1 so that we can turn this into a valid PESP-constraint. Then we get

$$((r_t)_i)_1 = ((r_{t'})_{i'})_1 \; \forall i, i' = q, \ldots, p \implies$$
$$\left[\frac{1}{n}60 - m, \;\; \frac{n-1}{n}60 \; + m\right]_{60} \in a(\varepsilon_q^t, \varepsilon_q^{t'}) \text{ and}$$
$$\left[\frac{k}{n}60 - m, \frac{k+n-1}{n}60 + m\right]_{60} \in a(\varepsilon_q^t, \varepsilon_q^{t'}) \;\; \forall k = 2, \ldots, n-1.$$

  This means that the margin m must meet that inequality.

  We make a distinction between Frequency constraints on a single train series and Frequency constraints on multiple train series. Just like in Section 1.4, it happens sometimes that different train series have part of their route in common. We can then enforce Frequency constraints on those trains using their combined frequency. Therefore if the set $\mathcal{F}$ contains trains of more than one train series, we say that we employ a *Shared Frequency* (SF) constraint on that set of trains, as all train series in the group then share that frequency constraint. If, on the other hand, $\mathcal{F}$ only contains trains of one particular train series, we call this a *Particular Frequency* (PF) constraint.

To illustrate why Frequency constraints allow for a union of multiple intervals, consider a set $\mathcal{F} := \{t_1, t_2, t_3, t_4\}$ of 4 trains. Remember that the goal of a frequency constraint is to make any pair of consecutive train departures $d(t), d(t')$ in the set have $15 - m$ to $15 + m$ minutes between them. However, the order of the trains in the set $\mathcal{F}$ is not yet known beforehand. If we were to have some order

$$0 \leq d(t_i) < d(t_j) < d(t_k) < d(t_l) \leq 59,$$

then for all consecutive pairs to have at least $15 - m$ and at most $15 + m$ minutes between them, non-consecutive pairs will be forced to have either $[30 - m, 30 + m]$ or $[45 - m, 45 + m]$ minutes between them. Therefore we allow any pair to have an interval between them that lies somewhere in

$$[15 - m, 15 + m] \cup [30 - m, 30 + m] \cup [45 - m, 45 + m],$$

which coincides with the intervals described above.

The other type of comfort constraints is Transfer constraints.

- **T-constraints.** When it is possible to transfer from one train series to another, we can schedule just the right amount of time (not too short, but also not too long) between the arrival and departure of the respective train series. This is enforced in a *T-constraint* or Transfer constraint.

Since we do not incorporate T-constraints in our model, we will leave it at this short mention and refer the reader to Van der Knaap's thesis for more information on them [11].

Let us look at an example that shows how these constraints are implemented on a line planning and a network. We refer to Example 2.2.1.1, and see which constraints need to be implemented on this network.

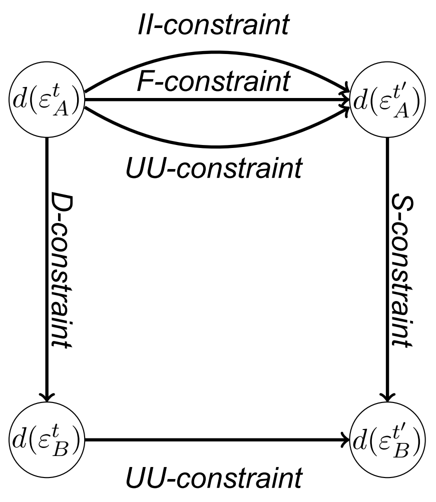
Example 2.2.1.1 (Continued)**.** We will first write the trip-defining constraints. We have train t that drives to station B without stopping, hence this needs a Drive constraint, and we have train t' that drives to station B with a stop. Therefore this needs a Stopping constraint.

To assemble the safety constraints it is important to check where the two trains t and t' could interfere with each other. They could do this on all stage points except C, hence we need an Out-Out constraint for the trains leaving station A and B, and an In-In constraint for the trains arriving at station B. Note that we do not need to impose any constraints of the In-Out, Out-In or Frontal Collision kind because the trains in this small example only travel in one direction, hence they never move in opposing directions.

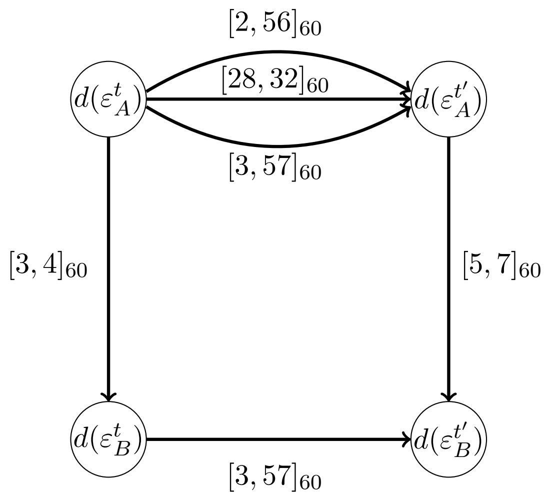
Lastly, a comfort constraint: since we want to ask the trains to leave their starting station in a regular frequency we impose a Frequency constraint on the trains leaving from station A.

These constraints are portrayed in the multigraph of Figure 2.5a, where all periodic events are portrayed as nodes and all constraints as directed edges. Note that a directed edge between two periodic events ε and ε' means that the constraint is on the ordered pair $(d(\varepsilon), d(\varepsilon'))$, and hence of the form $d(\varepsilon) - d(\varepsilon') \in [l, u]_{60}$.

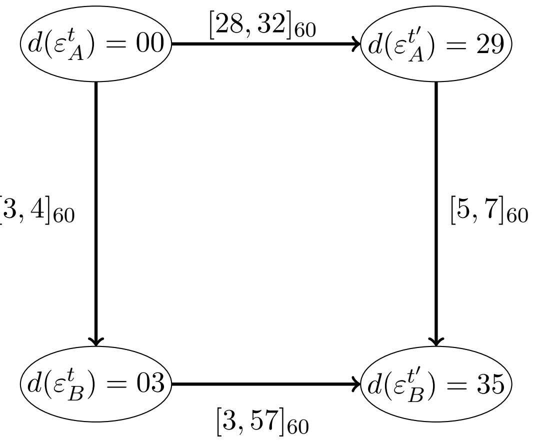
Below, the numerical constraints are stated. They can also be found in Figure 2.5b. Note that although between $d(\varepsilon_A^t)$ and $d(\varepsilon_A^{t'})$ there are three constraints, they unite into one more restricted intersection. This also means that between any pair there will be at most one constraint on the time difference between them.



(a) Types of the constraints



(b) Intervals of the constraints


$d(\varepsilon_A^t) = 00$
$[28, 32]_{60}$
$d(\varepsilon_A^{t'}) = 29$
$[3, 4]_{60}$
$[5, 7]_{60}$
$d(\varepsilon_B^t) = 03$
$d(\varepsilon_B^{t'}) = 35$
$[3, 57]_{60}$

(c) Solution for the given constraints displayed in the nodes

Figure 2.5: The directed multigraph of the PEN described in Example 2.2.1.1 in threefold: the types of constraints, the intervals of the constraints and a possible solution to the PESP. In each graph, the nodes portray the periodic events and the edges portray ordered constraints.

$$\begin{aligned}
&d(\varepsilon_A^t) - d(\varepsilon_A^{t'}) \in [2,56]_{60} \cap [28,32]_{60} \cap [3,57]_{60} \equiv [28,32]_{60}\\
&d(\varepsilon_A^t) - d(\varepsilon_B^t) \in [3,4]_{60}\\
&d(\varepsilon_A^{t'}) - d(\varepsilon_B^{t'}) \in [5,7]_{60}\\
&d(\varepsilon_B^t) - d(\varepsilon_B^{t'}) \in [3,57]_{60}
\end{aligned}$$

A solution to the PESP of Example 2.2.1.1 is the following time assignment:

$$\begin{aligned}
d(\varepsilon_A^t) &= 00;\\
d(\varepsilon_A^{t'}) &= 29;\\
d(\varepsilon_B^t) &= 03;\\
d(\varepsilon_B^{t'}) &= 35.
\end{aligned}$$

In Figure 2.5c we can see that this timetable indeed meets all of the constraints.

2.3. Satisfiability

The problem Satisfiability is integral to the model used in this thesis, and therefore a short introduction to the problem is useful. SAT in its basis is a logical boolean feasibility problem and it employs the definitions outlined below, which are taken from Grossmann [6]. We will illustrate the definitions by means of a simplified version of our SAT-variables.

Definition 2.3.0.1 (Alphabet of propositional logic)**.** The *alphabet of propositional logic* Σ_{SAT} consists of a countably infinite set of propositional variables $\mathcal{R} = \{p_1, p_2, \ldots\}$, the brackets "(" and ")" and the set of connectives $\{\wedge, \vee, \neg\}$. Here, $\wedge$, $\vee$, and $\neg$ signify the logical connectives and, or, and not, respectively. Therefore the $\wedge$ and $\vee$ connectives are binary, whereas the $\neg$ connective is unary.

Definition 2.3.0.2 (Propositional formula)**.** A string F, which consists solely of letters of the alphabet Σ_{SAT} is called a *propositional formula* if it fullfills one of the following properties:

1. if $F = p$, then $p \in \mathcal{R}$;
2. if $F = \neg G$, then G is a propositional formula ;
3. if $\circ$ is a binary connective and $F = (G \circ H)$, then G and H are propositional formulas.

Example 2.3.0.1. Let train t depart from a given station. We make a boolean variable that dictates whether the train departs before :30 or after. To this end we introduce the variable $q_{d(t),i}$ which indicates:

$$q_{d(t),30} := d(t) \leq 30,$$

so it has the meaning that train t departs *before* or exactly at the 30-minute mark. Since it is a boolean variable, it can be true or false. If it is false it says that train t leaves *after* :30, but if it is true it indicates that train leaves before or at :30. Let

$$\Sigma_{SAT} := \{q_{d(t),30}, \neg, \vee, \wedge, \text{“(”}, \text{“)”}\}.$$

With this alphabet we can easily make a propositional formula that dictates train t should *not* leave before or at :30, but that it instead should leave *after* the :30 mark:

$$\neg q_{d(t),30} \Leftrightarrow d(t) > 30.$$

Let $\mathcal{L}(\Sigma_{SAT})$ denote the smallest set that contains each propositional formula under the alphabet Σ_{SAT} (i.e., each propositional formula $F \in \mathcal{L}(\Sigma_{SAT})$ is unique).

Definition 2.3.0.3 (Literal)**.** A propositional formula $L \in \mathcal{L}(\Sigma_{SAT})$ is called a *literal* if $L = p$ or $L = \neg p$ with $p \in \mathcal{R}$.

Definition 2.3.0.4 (Clause)**.** A propositional formula $C \in \mathcal{L}(\Sigma_{SAT})$ is called a *clause* if it is a disjunction of literals. Therefore,

$$C = (\ldots(L_1 \vee L_2) \vee \ldots) \vee L_n),$$

where $n \geq 0$ and $L_i : i \in \{1, \ldots, n\}$ are literals, is a clause. The clause C may also be written as $C = [L_1, \ldots, L_n]$.

Example 2.3.0.1 (Continued)**.** We expand our alphabet by a new variable that dictates whether the departure of train t happens until the 45-minute mark or after:

$$q_{d(t),45},$$

such that Σ_{SAT} becomes

$$\Sigma_{SAT} := \{q_{d(t),30}, q_{d(t),45}, \neg, \vee, \wedge, \text{“(”}, \text{“)”}\}.$$

Now we can construct the clause

$$q_{d(t),30} \vee q_{d(t),45},$$

which states that the departure must either happen before the 30-minute mark *or* before the 45-minute mark. This means that if this clause is satisfied and hence one (or both) of the literals is, this train is allowed to depart until :30 or until :45.

Definition 2.3.0.5 (Conjunctive Normal Form)**.** A propositional formula $F \in \mathcal{L}(\Sigma_{SAT})$ is in *conjunctive normal form* (short: CNF) if it is a conjunction of clauses. Therefore,

$$F = (C_1 \wedge C_2 \ldots \wedge C_m),$$

where $m \geq 0$ and $C_i : i \in \{1, \ldots, m\}$ are clauses, is in conjunctive normal form.

Most SAT-solvers intake problems in Conjunctive Normal Form. Given an interpretation, this form also allows us to directly point at clauses that are not satisfied under the interpretation, making it easier to isolate problems.

Example 2.3.0.1 (Continued)**.** Using CNF we can write constraints on the departure time of the train that must both hold. We add one more variable to our alphabet,

$$q_{d(t),15} \Leftrightarrow d(t) \leq 15,$$

and write the following conjunction of clauses:

$$\left(\neg q_{d(t),15} \vee \neg q_{d(t),30}\right) \wedge \left(q_{d(t),30} \vee q_{d(t),45},\right)$$

which is to say that the departure may happen after :15 or after :30 *and* it must either happen before :30 or before :45. Hence the departure should be scheduled as such:

$$d(t) > 15 \text{ or } d(t) > 30$$

$$d(t) \leq 30 \text{ or } d(t) \leq 45.$$

Definition 2.3.0.6 (Interpretation)**.** Let $F \in \mathcal{L}(\Sigma_{SAT})$ be a propositional formula. Then the mapping

$$I : \mathcal{L}(\Sigma_{SAT}) \to \{\text{true}, \text{false}\}$$

$$I(F) = w$$

is called an *Interpretation* with $w \in \{\text{true}, \text{false}\}$.

Definition 2.3.0.7. A propositional formula F is *satisfiable* if there exists an interpretation I such that $I(F) = \text{true}$. A propositional formula F is *unsatisfiable* if $I(F) = \text{false}$ for all interpretations I.

Example 2.3.0.1 (Continued)**.** An interpretation that would satisfy our conjunction is the following:

$$\neg q_{d(t),15} = \text{true}, \ \neg q_{d(t),30} = \text{true}, \ q_{d(t),30} = \text{false}, \ q_{d(t),45} = \text{true},$$

because then

$$d(t) > 15, d(t) > 30, d(t) > 30, d(t) \leq 45,$$

and a solution to this would be $30 < d(t) \leq 45$.

Since at least one interpretation exists, the propositional formula

$$F = \left(\neg q_{d(t),15} \vee \neg q_{d(t),30}\right) \wedge \left(q_{d(t),30} \vee q_{d(t),45},\right)$$

is satisfiable.

Definition 2.3.0.8. A literal L is satisfied under an interpretation I if

1. $L = p \in \mathcal{R}$ and $I(p) =$ true or
2. $L = \neg p, p \in \mathcal{R}$ and $I(p) =$ false.

Definition 2.3.0.9. A clause $C = [L_1, \dots, L_n]$ is satisfied under an interpretation I if at least one literal $L_i : i \in \{1, \dots, n\}$ is satisfied under I.

Example 2.3.0.1 (Continued)**.** If we go back to our conjunction of clauses:

$$\begin{cases} \neg q_{d(t),15} \vee \neg q_{d(t),45} \\ q_{d(t),30} \vee \neg q_{d(t),45}, \end{cases}$$

Our interpretation states $I(q_{d(t),15}) =$ false and $I(q_{d(t),30}) =$ false but $I(q_{d(t),45}) =$ true. In this case, we satisfy the literals $\neg q_{d(t),15}$, $\neg q_{d(t),30}$ and $q_{d(t),45}$, but not the literal $q_{d(t),30}$. Furthermore, since both clauses have at least one literal satisfied under I, we indeed satisfy both clauses.

Definition 2.3.0.10. A propositional formula in CNF $F = \langle C_1, \dots, C_m \rangle$ is satisfied under an interpretation I if all clauses $C_i : i \in \{1, \dots, m\}$ are satisfied under I. In this case we write

$$I \models F.$$

Example 2.3.0.1 (Continued)**.** Since our formula F is in CNF and all clauses are satisfied under $I(q_{d(t),15}; q_{d(t),30}; q_{d(t),45}) = (\text{false}; \text{false}; \text{true})$, F is satisfied under the interpretation I.

We are now ready to formulate the SAT-problem, that consists of trying to find an interpretation (a variable assignment) that renders an entire propositional formula true.

Definition 2.3.0.11 (SAT-problem)**.** Let $F \in \mathcal{L}(\Sigma_{SAT})$ be a propositional formula in CNF. Decide whether

1. F is unsatisfiable, or
2. F is satisfiable. If there exists an interpretation I such that $I(F) =$ true, return I.

Example 2.3.0.1 (Continued)**.** If we were to have the following clauses:

$$\begin{cases} \neg q_{d(t),30} \vee q_{d(t),45} & (2.8) \\ \neg q_{d(t),30} \vee \neg q_{d(t),45} & (2.9) \\ q_{d(t),30} & (2.10) \end{cases}$$

the formula would not be satisfied under I. In fact, there is no interpretation possible at all.

We must have that $q_{d(t),30} =$ true if we want to satisfy clause 2.10. However, to satisfy clause 2.9 at least one of its literals must be true. Therefore $\neg q_{d(t),45}$ must be true, but then then $q_{d(t),45}$ must be false. Then we fail to satisfy clause 2.8.

The instance above is a simple example of the case that no feasible timetable is possible. However, these clauses are not all-encompassing when it comes to train scheduling. For example, there is no clause that states that the train cannot depart after the 59th minute, or that it must depart no earlier than the 0th minute. Fortunately, there is an intricate method of enforcing these time limits through SAT-variables, as will be outlined in Section 2.4 below.

2.4. Encoding PESP to SAT

In order to solve PESP as a SAT-instance, the PESP-instance must be transformed into an equivalent instance of SAT. In the model, this happens through ordered encoding.

2.4.1. Encoding PESP variables

In the method of order encoding by Grossmann [6], PESP variables get translated into SAT-variables by creating an equivalent set of SAT-variables: a set that consists of one boolean variable for each minute in an hour. This boolean variable then indicates whether or not the PESP-variable is below or above that minute. These boolean variables are denoted as $q_{x,i}$, which for x a periodic event and for all $i = 0, 1, \dots, 59$ indicates the following:

$$q_{x,i} = \text{true} \iff x \leq i.$$

We directly take over the definition from Grossmann himself:

Definition 2.4.1.1 (Order Encoding Function for Variables of a Finite Domain)**.** Let $x \in \mathcal{X}$ be a variable with $dom(x) = [l, u] \subset \mathbb{N}$ and $\mathcal{X}$ the variable space. Then the function

$$\text{encode\_ordered} : \mathcal{X} \to \mathcal{L}(\Sigma_{SAT})$$

is the order encoding function for a variable of the variable space $\mathcal{X}$ with

$$\text{encode\_ordered} : x \mapsto \bigwedge_{i \in [l+1, u-1]} \neg q_{x,i-1} \vee q_{x,i}$$

with $\forall i \in [l, u-1] : q_{x,i} \in \mathbb{R}$.

This order encoding results in a conjunction of clauses, therefore it is in CNF and complying with the regular format for SAT.

In this case any interpretation I will have one of the following two cases:

$$I(q_{x,i}) = \text{true} \iff x \leq i$$
$$I(q_{x,i}) = \text{false} \iff x \not\leq i$$

Given such an interpretation I, we can extract the value of the original PESP-variable as follows:

Definition 2.4.1.2 (Extracting Value of Interpretation of Order Encoded Variable)**.** Let x be a variable with $dom(x) = D = [l, u] \subset \mathbb{N}$ and I an interpretation, such that

$$I \models encode\_ordered(x).$$

Then there exists a $k \in [l, u]$ with

$$x = \xi_x(I) = \begin{cases} k \text{ if } k \in [l, u-1] : q^I_{x,k-1} = \text{false} \wedge q^I_{x,k} = \text{true} \\ u \text{ if } k = u : q^I_{x,u-1} = \text{false} \end{cases}$$

where $\xi_x(I) : I \mapsto x$ is the function that gets from a given interpretation I the value of the variable x.

Example 2.4.1.1. Let ε be a periodic event that can depart at times $0, 1, 2, 3, 4, 5, 6,$ such that $d(\varepsilon) \in [0, 1, 2, 3, 4, 5, 6]$. Then the PESP-variable we will encode is $d(\varepsilon)$. By definition 2.4.1.1 it gets order encoded by the conjunction of the following clauses:

$$\neg q_{d(\varepsilon),0} \vee q_{d(\varepsilon),1};$$
$$\neg q_{d(\varepsilon),1} \vee q_{d(\varepsilon),2};$$
$$\neg q_{d(\varepsilon),2} \vee q_{d(\varepsilon),3};$$
$$\neg q_{d(\varepsilon),3} \vee q_{d(\varepsilon),4};$$
$$\neg q_{d(\varepsilon),4} \vee q_{d(\varepsilon),5},$$

which is equivalent to

$$\neg d(\varepsilon) \leq 0 \vee d(\varepsilon) \leq 1;$$
$$\neg d(\varepsilon) \leq 1 \vee d(\varepsilon) \leq 2;$$
$$\neg d(\varepsilon) \leq 2 \vee d(\varepsilon) \leq 3;$$
$$\neg d(\varepsilon) \leq 3 \vee d(\varepsilon) \leq 4;$$
$$\neg d(\varepsilon) \leq 4 \vee d(\varepsilon) \leq 5,$$

which is in turn equivalent to

$$d(\varepsilon) > 0 \vee d(\varepsilon) \leq 1; \tag{2.11}$$
$$d(\varepsilon) > 1 \vee d(\varepsilon) \leq 2; \tag{2.12}$$
$$d(\varepsilon) > 2 \vee d(\varepsilon) \leq 3; \tag{2.13}$$
$$d(\varepsilon) > 3 \vee d(\varepsilon) \leq 4; \tag{2.14}$$
$$d(\varepsilon) > 4 \vee d(\varepsilon) \leq 5. \tag{2.15}$$

Remember that since this is a conjunction of clauses SAT needs to satisfy all clauses, so for each clause at least one of the literals must be true. In the case that both literals are true, the PESP-variable takes exactly the value indicated by these literals. We illustrate that only one clause can have this property, and therefore if *encode_ordered($d(\varepsilon)$)* is true, $d(\varepsilon)$ takes a unique value. If for example an interpretation satisfies both literals in clause 2.13, we find

$$d(\varepsilon) > 2 \land d(\varepsilon) \leq 3 \quad \Rightarrow \quad d(\varepsilon) = 3.$$

Now, because $d(\varepsilon) > 2$, we know that the second literals in clauses 2.11 and 2.12 are false, and that the first literals in the clauses are true. Similarly, we know that the clauses 2.14 and 2.15 both have their first literal falsified and their second literal satisfied.

Hence there is only one clause that can have both literals be true, and this illustrates that the order-encoded function causes all PESP-variables to have a unique value. This is important because we cannot have trains departing at multiple times.

We will now prove this more formally in Theorem 2.4.1.1.

Theorem 2.4.1.1. If $encode\_ordered(x)$ is true, there is a unique value $k \in [l, u]$ that x takes such that

$$\forall i \in [l, k-1],\ q^I_{x,i} = \text{false},$$
$$\forall i' \in [k, u-1],\ q^I_{x,i'} = \text{true}.$$

Proof. If $k \in [l, u-1]$, $q^I_{x,i}$ is defined by the truth assignment in Definition 2.4.1.1, and so

$$q^I_{x,k-1} = \text{false and } q^I_{x,k} = \text{true},$$

so $x \not\leq k-1$ and $x \leq k$. Therefore $x = k$.

If $k = u$, then by Definition 2.4.1.2 we have that $\forall i \in [l, u-1]$,

$$q^I_{x,i} = \text{false},$$

and by definition of $q_{x,i}$,

$$q^I_{x,u-1} = \text{false} \Leftrightarrow x \not\leq u-1.$$

Now since $x \in [l, u]$, this means $x = u = k$. □

We can therefore easily extract the value of x, given that a truth assignment I is valid.

Now for a railway timetabling periodic event network $\mathcal{N} = (\mathcal{V}, T, a)$ the variables are $d(\varepsilon^t_i) \in \mathcal{V}$ for $t = 1, \ldots, N$ and $i = 1, \ldots, s_t - 1$. The propositional formula that encodes these variables then becomes:

$$\Omega^{\mathcal{N}}_{ordered} := \bigwedge_{t=1,\ldots,N, i=1,\ldots,s_t-1} encode\_ordered(d(\varepsilon^t_i)).$$

Since this is again a conjunction of clauses, it is in CNF and ready to be used as input for SAT.

2.4.2. Encoding PESP constraints

Remember that PESP-constraints are of the form

$$d(\varepsilon) - d(\varepsilon') \in [l, u]_{60}.$$

Example 2.2.1.1 (Continued)**.** To illustrate the process of translating a PESP-constraint into a SAT-constraint we perform a transformation on one of the constraints from Example 2.2.1.1 between

$d(\varepsilon^t_A)$ and $d(\varepsilon^{t'}_A)$, namely the following:

$$d(\varepsilon^t_A) - d(\varepsilon^{t'}_A) \in [28, 32]_{60}. \tag{2.16}$$

Because $0 \leq d(\varepsilon^t_A), d(\varepsilon^{t'}_A) \leq 59$, this constraint can be visualized on a 59×59 grid, where in the graph $d(\varepsilon^t_A)$ is portrayed on the x-axis and $d(\varepsilon^{t'}_A)$ on the y-axis, such as in Figure 2.6. In this Figure the green parts are the feasible regions and therefore the regions where $d(\varepsilon) - d(\varepsilon') \in [28, 32]_{60}$, whereas the red parts are the infeasible regions where the time difference between the two periodic events does *not* lie within this interval. Note that for clarity the entire area is filled up, even though since our variables can only take integer values the only feasible points are actually the integers.

To encode this constraint, we describe the infeasible regions in the form of line segments by use of the SAT-variables that we just created. These line segments are formatted in red in the figure.

We want to describe all the red lines in the infeasible region such that they form SAT-clauses, as the SAT-solver takes a conjunction of clauses as input. We now describe only the blue line segment in Figure 2.6,

$$\left(d(\varepsilon^t_A), d(\varepsilon^{t'}_A)\right) \in 20 \times [0, 47].$$

To enforce that the difference between the pair of periodic events is *not* allowed to be on this line segment, we need to ask that

$$\nexists \left(d(\varepsilon^t_A), d(\varepsilon^{t'}_A)\right) : d(\varepsilon^t_1) = 20, 0 \leq d(\varepsilon^t_1) \leq 47,$$

which is equivalent to

$$\begin{aligned}
&\neg \left((d(\varepsilon^t_A) \geq 20) \wedge (d(\varepsilon^t_A) \leq 20) \wedge (d(\varepsilon^{t'}_A) \geq 0) \wedge (d(\varepsilon^{t'}_A) \leq 47)\right) \\
\Leftrightarrow &\neg \left(\neg(d(\varepsilon^t_A) < 20) \wedge (d(\varepsilon^t_A) \leq 20) \wedge \neg(d(\varepsilon^{t'}_A) < 0) \wedge (d(\varepsilon^{t'}_A) \leq 47)\right) \\
\Leftrightarrow &\neg \left(\neg(d(\varepsilon^t_A) \leq 19) \wedge (d(\varepsilon^t_A) \leq 20) \wedge \neg(d(\varepsilon^{t'}_A) \leq -1) \wedge (d(\varepsilon^{t'}_A) \leq 47)\right) \\
\Leftrightarrow &\left((d(\varepsilon^t_A) \leq 19) \vee \neg(d(\varepsilon^t_A) \leq 20) \vee (d(\varepsilon^{t'}_A) \leq -1) \wedge \neg(d(\varepsilon^{t'}_A) \leq 47)\right) \\
\Leftrightarrow &\left(q_{d(\varepsilon^t_A)),19} \vee \neg q_{d(\varepsilon^t_A),20} \vee q_{d(\varepsilon^{t'}_A),-1} \vee \neg q_{d(\varepsilon^{t'}_A),47}\right)
\end{aligned}$$

Remember that $q_{d(\varepsilon),i}$ symbolizes $d(\varepsilon) \leq i$. Hence this line means that

$$d(\varepsilon^t_A) \leq 19 \text{ or } d(\varepsilon^t_A) \geq 21 \text{ or } d(\varepsilon^{t'}_A) \leq -1 \text{ or } d(\varepsilon^{t'}_A) \geq 48,$$

which allows the difference between the two periodic events to be anything *except* the blue line segment in Figure 2.6.

To accurately formulate this entire constraint we would need to describe all 26+60+27 = 113 infeasible line segments in that way. In general, an infeasible line segment of a PESP constraint can be described in the following way.

Definition 2.4.2.1. Let $\varepsilon, \varepsilon'$ be periodic events and let $c \in a(\varepsilon, \varepsilon')$ be a PESP constraint. Then

$$\zeta(\varepsilon, \varepsilon', c) = \{\{x\} \times [y_1, y_2] | x \in dom(d(\varepsilon)); y_1, y_2 \in dom(d(\varepsilon'));\\ y - x \in c \forall y \in [y_1, y_2]; y - x \notin c \text{ for } y = y_1 - 1, y_2 + 1\}$$

is the set of all infeasible line segments imposed by constraint c between the events ε and ε'.

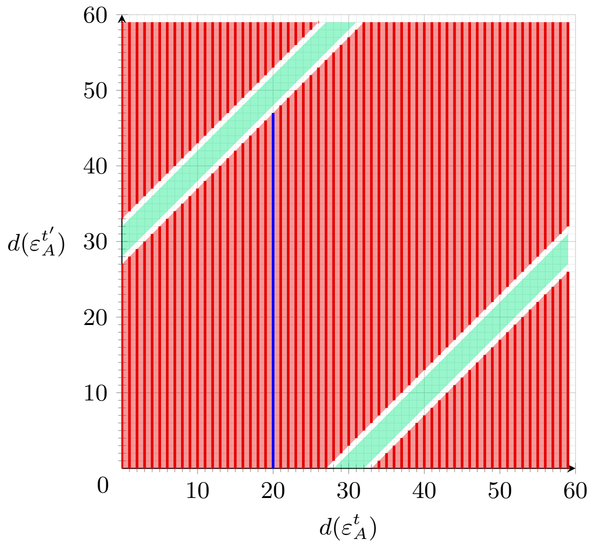


Figure 2.6: Feasible (green) and infeasible (red) regions of PESP constraint 2.16: $d(\varepsilon_A^t) - d(\varepsilon_A^{t'}) \in [28, 32]_{60}$. The line segment $\left(d(\varepsilon_A^t), d(\varepsilon_A^{t'})\right) \in 20 \times [0, 47]$ that makes up part of the constraint and is translated into a SAT-clause in Example 2.2.1.1 is emphasized in blue.

We can then formulate the clauses that encode all line segments of a constraint $c \in a(\varepsilon_i, \varepsilon_j)$ as such:

Definition 2.4.2.2. Let ε, ε' be two periodic events, and let $c \in a(\varepsilon, \varepsilon')$ be a PESP constraint. Then

$$encode\_ordered\_con : \mathcal{V} \times \mathcal{V} \times \mathcal{I}_T \to \mathcal{L}(\Sigma_{SAT})$$
$$(\varepsilon, \varepsilon', c) \mapsto \bigwedge_{\{x\} \times [y_1, y_2] \in \zeta(\varepsilon, \varepsilon', c)} \left(q_{d(\varepsilon), x-1} \vee \neg q_{d(\varepsilon), x} \vee q_{d(\varepsilon'), y_1 - 1} \vee \neg q_{d(\varepsilon'), y_2}\right)$$

is the order encoding function of PESP constraint c.

Now, given a railway timetabling periodic event network $\mathcal{N} = (\mathcal{V}, T, a)$, the propositional formula which encodes all PESP constraints is

$$\Psi_{ordered}^{\mathcal{N}} := \wedge_{t, t', i, i'} \wedge_{c \in a(\varepsilon_i^t, \varepsilon_{i'}^{t'})} encode\_ordered\_con(\varepsilon_i^t, \varepsilon_{i'}^{t'}, c),$$

where $t, t' = 1, \ldots, N$, $i = 1, \ldots, s_t - 1$ and $i' = 1, \ldots, s_{t'} - 1$. This formula is in CNF.

2.4.3. Encoding PESP

The entire propositional formula that encodes the variables and constraints of PESP can then be formulated as follows.

Definition 2.4.3.1. Let $\mathcal{N} = (\mathcal{V}, T, a)$ be a Periodic Event Network. Then

$$\Omega^{\mathcal{N}}_{ordered} \wedge \Psi^{\mathcal{N}}_{ordered}$$

is the propositional formula that encodes $\mathcal{N}$.

2.5. Formulation of Van der Knaap's model

It is important to give credit to Vollebergh as her model played an important part in that of Van der Knaap. In fact, both models follow the same concept: given a train network, a line planning and routes for the trains, check if a solution is available through transforming the problem to a Satisfiability instance, then calling a SAT-solver. If no solution is available, add the next right-most route options until either a solution does become available, or all route options have been exhausted. That said, since Van der Knaap's model is the most recent version and we will therefore use her model as a start to ours, we will refer to her model only from now on.

Van der Knaap's model is based on the Periodic Event Scheduling Problem detailed in Chapter 2.1 and it translates the problem to a SAT-instance as described in Chapter 2.4. To be able to employ flexible track options, the model makes use of a modified version of PESP called OPESP . This acronym stands for Open Periodic Event Scheduling Problem and it entails that each periodic event can 'happen' in a multitude of ways. For our problem this means that in the periodic event of a train leaving a certain stage point, it can do so on a multitude of track options. Since this is not of immediate importance to our model, we will leave the explanation as short as that. The interested reader is referred to the thesis of Van der Knaap [11], in which a detailed account of OPESP is given.

Van der Knaap has improved Vollebergh's model by incorporating the extraction of information from the SAT-solver to find promising extra route options. Her model therefore adds fewer route options to the problem than Vollebergh's did, thereby eliminating redundant variables and constraints. This information is found by extracting a minimal unsatisfiable subformula (MUS) from the SAT-instance. A MUS is a special type of an *unsatisfiable core* and the ability to find one is a property of the SAT-solver.

Definition 2.5.0.1 (Unsatisfiable core)**.** Given a formula in conjunctive normal form F, ϕ is an *unsatisfiable core* for F if ϕ is an unsatisfiable formula in CNF and $\phi \subseteq F$.

Remember that conjunctive normal form (CNF) indicates a conjunction of clauses, where a conjunction is a juxtaposition of literals connected by and-operators. This allows for a SAT-solver to view any one clause as a separate restriction. Unsatisfiable cores can be chosen in such a way that removing only one clause results in a satisfiable subset, hence gaining feasibilty within that particular subset. Such an unsatisfiable core is called a minimal unsatisfiable core.

Definition 2.5.0.2 (Minimal unsatisfiable core)**.** Let ϕ be an unsatisfiable core of the formula F which is in CNF. Then ϕ is a *minimal unsatisfiable core* if $\phi\backslash\{C\}$ is satisfiable for all $C \in \phi$. This is also called a *minimal unsatisfiable subformula* (MUS).

Hence by deriving a MUS Van der Knaap's model is able to spot sets of conflicting clauses within the SAT-instance. Recall that these clauses represent constraints from the PESP, and any constraint is always enforced between two different departures of trains from certain stage points. Therefore any clause can be traced back to the departures it pertains to. This allows the model to add extra track options in places that seem troubling: stations or train series that are often involved in the MUS are likely to be causing the conflicts. It adds these track options near those stations or along the routes of those train series.

Next to that, rather than starting out with the right-most track options for all trains, her model starts with track routes found by a dynamic programming algorithm. This further increases the chances of quickly finding the track options that are necessary to be able to adhere to all constraints. Summarized in pseudocode, Van der Knaap's model operates in the following way:

```
Given: Network, line plan
Returns: A timetable or a statement that no timetable can be found before one of the
         stopping criteria is met
Initialize instance;
Find routes within the network for all trains;
while no timetable is found & the maximal number of attempts has not been reached
  do
    Translate problem from OPESP to SAT;
    Give problem to SAT solver;
    if problem is unsatisfiable then
        Extract information from MUS;
        Find new track options to add to the problem;
        if no new track options are found then
            Exit while loop;
        end
    end
end
if a timetable was found then
    Extract timetable from outcome SAT solver;
    Print timetable;
end
```

Algorithm 1: findTimetable by Van der Knaap[11]. The algorithm has been modified slightly to be understandable within the given context.

We now have an algorithm in place that, given a set of constraints, is able to find routes and times for all trains such that all constraints are met *if such a schedule exists*. Because our goal is to solve an instance of which we know beforehand that there is no schedule that meets all constraints due to comfort constraints, we can simply omit these comfort constraints from the list of constraints, only leaving in trip-defining constraints and safety constraints.

3

Using Weighted Partial maxSAT to incorporate user wishes

3.1. The idea of the method

If for a certain network it is known that all trains ‘fit’ without violating any strictly required constraints, then a train timetable can be construed in which no collisions occur and all trains maintain a safe distance from each other, i.e. a feasible timetable. The next goal is then to add as many comfort constraints as possible without affecting the feasibility of the problem, although in the worst case scenario ‘as many as possible’ may be zero. This may result in a timetable that sets all trains of a certain train series to leave in the first fifteen minutes or so, but it will result in a timetable nonetheless. This idea is illustrated in Figure 3.1.

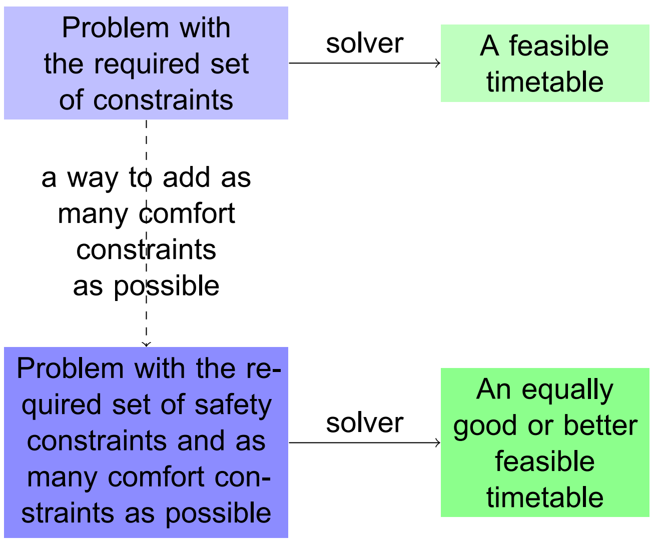

Problem with the required set of constraints
solver
A feasible timetable
a way to add as many comfort constraints as possible
Problem with the required set of safety constraints and as many comfort constraints as possible
solver
An equally good or better feasible timetable

Figure 3.1: Diagram illustrating the idea of incorporating comfort constraints while maintaining feasibility. If a feasible timetable can be found for a strictly required set of constraints, then a feasible timetable can always be found if the goal is to add ‘as many’ comfort constraints ‘as possible’. If adding comfort constraints is in fact possible, this will result in a better feasible timetable. Note that for constructing a possibly better feasible timetable, the feasible timetable in the top is not directly used. Only the knowledge that such a timetable exists, is needed.

The difficulty in applying this idea lies in two parts.

First of all, the concept of 'as many as possible' is not as concrete as it sounds. While it is true that the inclusion of any comfort constraint makes the timetable more convenient for travelers, the belief that any two different comfort constraints will have the exact same worth for the total traveler comfort is incorrect. Some constraints may benefit thousands of travelers, while others benefit only a few. In fact, an entire study could be devoted to finding out to what degree certain constraints advantage travelers, but in this report we will limit this to the small Appendix A. Instead, we will simply allow some comfort constraints to weigh heavier than others.

Secondly, we need a model that finds a feasible solution that adheres to all safety constraints. Fortunately that part has been taken care of for us, as my predecessors Vollebergh [20] and Van der Knaap [11] have devised models that do precisely that.

Lastly, we need a proficient method to incorporate as many constraints as possible while taking into account their difference in importance. Such a method is described in Section 3.2.

3.2. (Weighted Partial) maxSAT

In solving a SAT-problem, it is not always possible to find an interpretation that renders all SAT-clauses true. However, it is better to have a solution that violates a minimal number of clauses than to have no solution at all. MaxSAT solvers do exactly that, as when given a SAT-problem their goal is to determine the maximum number of clauses that can be satisfied. Through solving this question they find such a maximal assignment. Because the aim of maxSAT is to find a solution that satisfies simply as many clauses as possible, it views all clauses as equally important. We would like to embody that for some clauses it is more important that they are satisfied than others though, and therefore we move on to the problem of Weighted maxSAT. In this problem, each clause is assigned a weight such as in the following definition from Ansótegui, Bonet and Levy [2].

Definition 3.2.0.1 (Weighted clause)**.** A *weighted clause* is a pair (C, w) where C is a clause and w is a natural number or infinity, indicating the penalty for falsifying C. A clause is called *hard* if the corresponding weight is infinity, otherwise the clause is called *soft*.

Therefore it is possible to distinguish clauses that are allowed to be violated (soft clauses) and clauses that must always be satisfied (hard clauses). The latter get weight infinity and hence assignments that do not satisfy hard clauses will not be proposed by the maxSAT solver. The combination of these hard and soft clauses is called a (weighted partial) maxSAT formula.

Definition 3.2.0.2. A *(weighted partial) maxSAT formula* is a set of weighted clauses

$$\phi = \{(C_1, w_1), \dots, (C_m, w_m), (C_{m+1}, \infty), \dots, (C_n, \infty)\}$$

where the first m clauses are soft and the last $(n - m)$ clauses are hard. Given a weighted partial maxSAT formula ϕ, we define the SAT formulas $\phi_{soft} = \{C_1, \dots, C_m\}$, $\phi_{hard} = \{C_{m+1}, \dots, C_n\}$ and $\phi_{plain} = \phi_{soft} \cup \phi_{hard}$.

Note that in the case that all clauses have weight 1 and the set of hard clauses is empty, a weighted partial maxSAT formula is just a maxSAT formula. Given an interpretation I, the *cost* of the maxSAT formula ϕ can be found by adding up all weights of clauses falsified by the interpretation. The problem of solving a weighted partial maxSAT formula is then finding an interpretation $I : \Sigma_{SAT} \to \{\text{false}, \text{true}\}$ such that the cost gets optimized as in Definition 3.2.0.3.

Definition 3.2.0.3 (Optimal cost and assignment)**.** Given a weighted partial maxSAT formula ϕ and an interpretation $I : \Sigma_{SAT} \to \{0, 1\}$, the *optimal cost* of a formula is the minimal cost of all its interpretations:

$$\text{optimal cost}(\phi) = \min \left\{ \sum_{(C_i, w_i) \in \phi, I(C_i) = 0} w_i \middle| I : \Sigma_{SAT} \to \{\text{false}, \text{true}\} \right\}$$

An *optimal assignment* is an assignment with optimal cost. A maxSAT formula is said to be satisfiable if it has cost zero.

Hence the optimal cost of a weighted partial maxSAT formula is the minimum of the sum of the weights of unsatisfied clauses, optimized over all possible interpretations. The *Weighted Partial maxSAT problem* for a weighted partial maxSAT formula ϕ is the problem of finding an optimal assignment which renders all hard clauses true, and falsifies a number of clauses such that the combined cost of the violated clauses is minimal. It is therefore possible to assign more importance to some clauses than to others, which is exactly what we want. We will set the weights of the trip-defining constraints and the safety constraints at infinity, so that these constraints will always be upheld.

3.2.1. The maxSAT solver

The maxSAT solver used in our experiments is part of the Sat4j library in Java and was built by D. Le Berre and A. Parrain [4]. In their maxSAT solver, a new variable per weighted soft clause is created which represents that that clause has been disobeyed. Then the sum of the weights multiplied by these indicator variables gets minimized. The solver uses linear search: it searches for new solutions (guessing solutions and checking them with a SAT-solver) and records when it finds a better one, only achieving new *upper* bounds [4]. It records these solutions by appending new hard clauses that prevent new solutions with a worse objective than the current optimal value. Therefore the solver does not have to try all solutions. Unfortunately, having no lower bound means that it takes longer to prove optimality, as the solver still simply has to check all remaining solutions.

The method described above shows that the maxSAT solver has to find exactly how many 'as many as possible' is, each time it tries a new solution, so it actually performs the bottom half of the diagram in Figure 3.1 many times to find the optimal solution.

3.3. Using maxSAT in combination with Van der Knaap's model

Now that it is clear what Van der Knaap's model entails in broad strokes and we have a tool to optimize the obedience of comfort constraints that we find important, we can go over the entire model, which we call maxSAT_s from now on.

First we use Van der Knaap's model to find out if a feasible timetable is at all possible for the given network and line planning. We ask her model to obey all constraints we deem hard – safety constraints, stopping constraints and driving constraints. If, perhaps after some rounds of adding possible track options, a timetable is found, this means that the network is equipped to sustain the trains in the line plan and all trains can safely make their trips without interfering with others.

We then create comfort constraints that we wish to implement upon the system. We first run a SAT solver to see if perhaps a solution is available already before moving on to the next step. When no solution is found, we assign the soft constraints weights. After transforming these constraints into equivalent SAT-clauses, we make a maxSAT file that contains all hard clauses that originated from Van der Knaap's model in addition to the soft clauses we just devised. The soft clauses get accompanied by their weights and the hard clauses get assigned weight infinity. Now the SAT solver is called upon and it tries to find an interpretation that falsifies a number of clauses such that the total weight of all unsatisfied clauses is as low as possible.

When the maxSAT solver is done optimizing or it has timed out, it returns a timetable that at least adheres to all hard constraints. Still it may be that Van der Knaap's model, that did not take into account any shared frequency constraints, gives a better solution. This could happen if the maxSAT solver has not yet reached optimality before it times out. In this case we assume Van der Knaap's solution as the final solution. Otherwise we assume the maxSAT solver's solution. The pseudo-code in Figure 2 summarizes our model.

```
Given: Network, line plan, routes for all trains, weights for soft constraints
Returns: A timetable
Initialize instance;
Run model RvdK without frequency constraints;
if model RvdK without frequency constraints is satisfiable then
    Write frequency constraints and translate them to SAT;
    Combine frequency constraints and constraints of RvdK;
    Try to solve as SAT instance;
    if problem is satisfiable then
        Return SAT-solution;
    end
    else
        Assign all constraints of RvdK maximal weight;
        Assign all frequency SAT constraints the defined weights;
        Combine soft and hard constraints into one maxSAT file;
        Run maxSAT solver until timeout is reached or solution is proved optimal;
        Return maxSAT solution;
        Compare maxSAT solution to RvdK solution and select the best;
        Extract timetable from the winning solution;
        Print timetable;
    end
end
else
    Print "Safety constraints or trip-defining constraints conflicting";
end
```

Algorithm 2: findTimetable

Violating clauses versus violating constraints with the maxSAT solver

When the maxSAT solver tries to find the best solution, it minimizes the total sum of weights of soft *clauses* violated and not that of the soft *constraints*, even though our goal is the latter. Fortunately, when a clause is violated, the constraint that originated that clause is violated and vice versa. Nonetheless, the two are not entirely equivalent. To understand why, we must revisit the way that the PESP frequency constraints get transformed into Satisfiability. Remember how frequency constraints are imposed in PESP:

Let $\mathcal{F}$ be a group of trains sharing part of their route, on which we impose frequency constraints. Let $d(\varepsilon_x^t)$ denote the departure time of train t from universal stage point x (by abuse of notation). For all $t, t' \in \mathcal{F}, t \neq t'$, we ask that

$$d(\varepsilon_x^{t'}) - d(\varepsilon_x^t) \in \bigcup_{k=1}^{n-1} \left[\frac{k}{n}60 - m, \frac{k}{n}60 + m\right]_{60}.$$

This means that a single frequency constraint is enforced by $\binom{|\mathcal{F}|}{2}$ subconstraints, one for each pair $(t, t') \in \mathcal{F}$. When one of the subconstraints is violated the entire frequency constraint is violated, though this does not necessarily mean that all is lost. Two trains t_1 and t_2 can have a too short amount of time between them, while train t_2 and t_3 still occur on perfect intervals.

Remember that because these subconstraints get turned into SAT-clauses, some 60+ clauses signify one subconstraint. Violating this subconstraint is equivalent to violating exactly one clause. This is due to our variable-defining clauses, which enforce that any periodic event happens at a unique minute in the hour as explained in Theorem 2.4.1.1. We illustrate this through the following example of a frequency subconstraint imposed on the two trains t and t':

$$d(t) - d(t') \in [18, 22]_{60} \cup [38, 42]_{60}. \tag{3.1}$$

Imagine that the maxSAT solver returns a solution in which this subconstraint is violated such that $(d(t), d(t')) = (10, 10)$ (see Figure 3.2). Because the PESP variables $d(t)$ and $d(t')$ both have one unique value, the respective departure times of train t and t' must lie on exactly one red line segment that signifies that this particular value is not allowed. It cannot lie on more line segments, because that would mean that $d(t)$ and $d(t')$ are assigned more than one value. Therefore if a subconstraint is violated, only one clause is violated and if the maxSAT solver returns a violated clause, this must be the only clause that originated from that subconstraint that got violated. Hence when the maxSAT solver solves the problem, a subconstraint getting violated adds to the sum of weights of violated constraints only one weight: that of the violated clause.

An unwanted property of PESP frequency constraints

When all PESP frequency subconstraints are met, the intervals between trains on which F-constraints are imposed are regular. But regular intervals do not necessarily meet all PESP F-subconstraints constraints. We can view this in the following example:

Example 3.3.0.1. We have a set $\mathcal{F} = \{t_1, t_2, t_3, t_4\}$ of four trains on which we want to enforce frequency constraints and we allow for a deviation of $m = 2$. PESP then asks the intervals between any pair of trains in the set to be within

$$[13, 17] \cup [28, 32] \cup [43, 47]. \tag{3.2}$$

If we set the trains to depart on

$$d(t_1) = 0;\ d(t_2) = 13;\ d(t_3) = 26;\ \text{and } d(t_4) = 43,$$

we obtain 13-13-17-17 intervals, so our intervals between consecutive trains do not deviate more than the allowed $m = 2$ from the perfect interval 15. However, the allowed intervals between

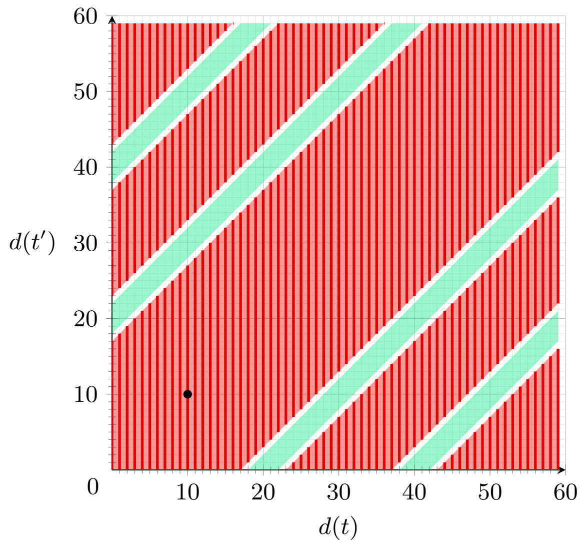


Figure 3.2: Value of $(d(t), d(t'))$ for which constraint 3.1: $d(t) - d(t') \in [18, 22]_{60} \cup [38, 42]_{60}$ is violated: $(d(t), d(t')) = (10, 10)$

non-consecutive trains *are* violated. For instance, between $d(t_1)$ and $d(t_3)$ there is an interval of 26 minutes, which is not in the intervals dictated in Equation 3.2.

In the case described above, PESP would deem such a solution infeasible, although it seems to be fine if we only look at the intervals. With the assumption that passengers arrive at a station randomly with a constant rate, we also see that if the trains were to leave with the intervals 13-17-13-17, which *is* allowed by PESP, train t_2 and t_3 are equally crowded as if they would have left in 13-13-17-17 intervals.

Even though this means that PESP cuts away some seemingly good solutions, it is still true that if all PESP F-constraints are met, the intervals are regular. This is not always so when it is allowed to satisfy some subconstraints while falsifying others such as with maxSAT. Then this unwanted property of PESP F-constraints can cause even more trouble, as is illustrated in the next example.

Example 3.3.0.2. $\mathcal{F} = \{t_1, t_2, t_3, t_4, t_5, t_6\}$. We could make the following two schedules:

$$d(t_1) = 0;\ d(t_2) = 18;\ d(t_3) = 20;\ d(t_4) = 38;\ d(t_5) = 40;\ d(t_6) = 58$$

$$\Leftrightarrow$$

$$\text{intervals } 18 - 2 - 18 - 2 - 18 - 2 \tag{3.3}$$

or

$$d(t_1) = 0;\ d(t_2) = 12;\ d(t_3) = 24;\ d(t_4) = 36;\ d(t_5) = 44;\ d(t_6) = 52$$

$$\Leftrightarrow$$

$$\text{intervals } 12 - 12 - 12 - 8 - 8 - 8 \tag{3.4}$$

Remember that in case of an F-constraint with frequency 6 the allowed bounds for any interval are: $[8, 12] \cup [18, 22] \cup [28, 32] \cup [38, 42] \cup [48, 52]$. Of course, option 3.4 seems much better and all intervals of consecutive trains meet the constraints, but because of the intervals of non-consecutive trains, option 3.4 fails to meet 6 subconstraints while option 3.3 only fails 3 of them.

As mentioned before, the reason that PESP operates in this way is because it does not know the orders of the trains beforehand, and therefore it is unfortunately hard to enforce PESP F-constraints in a different manner. Appendix B is dedicated to underlining the difference between optimizing the adherence to PESP F-constraints and the adherence to more intuitive frequency constraints.

Writing separate clauses
Van der Knaap's model efficiently clusters all constraints belonging to any one pair into a single constraint. Look back to Figure 2.5, where there are three constraints between periodic events $d(\varepsilon_A^t)$ and $d(\varepsilon_A^{t'})$. Because all constraints are hard (in Van der Knaap's model), the intersection of these constraints is –at least constraint-wise– the same as equipping the model with all of them separately. This has the result that between any two periodic events there is only one constraint. It is not entirely the same, because in our case we want to distinguish different (sub)constraints between one pair of events, and therefore we need to file all of our constraints separately. This causes more clauses to be written, as for any pair on which multiple frequency constraints are enforced, at least one more diagram such as in Figure 3.2 worth of clauses gets written.

3.3.1. Cutting away duplicate solutions by fixing the order of trains

It is likely that the maxSAT solver will take longer than the SAT solver. This is because the SAT solver only needs to find one solution or a proof that there is no solution, while the maxSAT solver needs to check many solutions. As there are at least in the order of a thousand extra clauses that the maxSAT solver is allowed to either satisfy or falsify[1], there are very many possible solutions and the maxSAT solver has to go over many of them to get to an optimal one.

One of the reasons that there are many solutions in PESP-problems in general is that there are many equivalent solutions. It is possible to indentify some obvious equivalent solutions. For instance, if a train series has trains departing three times an hour, PESP registers this as three different departures $d(t_1), d(t_2)$ and $d(t_3)$. Then among all conceivable timetables there are in total six possibilities from shuffling those three trains, even though they fulfill exactly the same role. The five duplicate solutions can be cut away by fixing the order of different trains within the same train series. We fix the order for all trains $t_i^k, i = 1, 2, \dots, n$ within a train series k with frequency n as follows:

$$d(t_1^k) < d(t_2^k) < \ldots < d(t_n^k).$$

This does not get rid of any unique solutions because within a train series all trains are interchangable. Hence this removes symmetries in the problem, making the solution space smaller and therefore curbing running time. Note that we can easily enforce an order between any two train departures $d(t)$ and $d(t')$ in SAT, where it gets enforced that $d(t) < d(t')$ as in Figure 3.3.

Moreover, it is possible to cut away equivalent solutions by presetting the order of trains in different train series on the same route, namely sets of trains on which we enforce shared frequency constraints. Unfortunately it is not evident that this does not cut away good or optimal solutions; it may well be that the optimal solution needs a threesome of trains to be in a particular order, such that enforcing an order different from that one cuts away this optimal solution. Nonetheless we will test if enforcing certain orders will speed up the maxSAT solver and if it will cause the solver to find equally good solutions within the same time frame or better solutions after an equal amount of running time. We will fix the order within groups of trains that have conflicting frequency constraints and we will consider three different orders:

[1]As is outlined in Section 4.2, even the smallest instance, Instance 1, causes an addition of 10+3+1 frequency subconstraints. Each frequency subconstraint results in at least roughly 90 line segments (see Figure 2.6), so 90 clauses. Since $14 \times 90 = 1260$, such a set of frequency constraints adds about 1000 clauses.

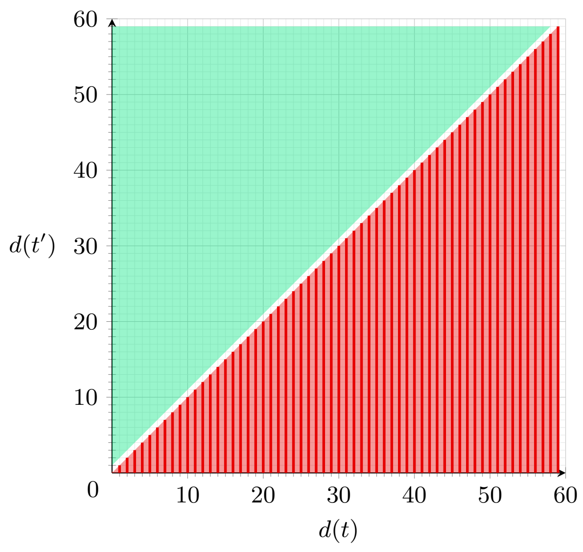


Figure 3.3: Illustration of the SAT-clauses that enforce that $d(t)$ must always happen before $d(t')$, such that $d(t) < d(t')$. Again, the green represents the feasible area and the red represents the infeasible area for the ordered pair $(d(t), d(t'))$.

1. The order inside the timetable created by Van der Knaap's model, including frequency constraints within time series;
2. The order inside the timetable created by maxSAT_s, which was found after running the maxSAT solver for 1 hour. Note that unlike the other orders this method iterates on itself, using previously found results. It therefore uses an additional hour, but it is still interesting to know whether the order found by maxSAT_s after running the solver for 1 hour is a 'good' order.
3. An alternating order. This order is devised by an 'educated guess'. It makes sense that for $k \in \mathbb{N}$ train series $t^1, t^2, \ldots, t^k$, with frequencies $n^1, n^2, \ldots, n^k \in \mathbb{N}$ respectively, enforcing an order such as

   $$d(t_1^1) < d(t_2^1) < \ldots < d(t_{n^1}^1) < d(t_1^2) < \ldots < d\left(t_{n^2}^2\right) < \ldots < d(t_1^k) < \ldots < d\left(t_{n^k}^k\right)$$

   will cause all trains of any one train series to depart closely after each other, leading to suboptimal solutions when the goal is to have all trains divided evenly across the hour. Therefore for $k \in \mathbb{N}$ train series $t^1, t^2, \ldots, t^k$, with frequencies $f^1, f^2, \ldots, f^k \in \mathbb{N}$ respectively, the alternating order will be

   $$d(t_1^1) < d(t_1^2) < \ldots < d(t_1^k) < d(t_2^1) < d(t_2^2) < \ldots < d(t_2^k) < \ldots < d(t_{n_1}^1) < d(t_{n_2}^2) < d(t_{n_k}^k).$$

The above fixed orders will be taken in by the model maxSAT_s, so it is in essence the same model. However, to avoid confusion we will call the model that has an order enforced over different train series on the same route *maxSAT_o* .

3.4. Performance measuring

Although it depends on the available time how good the solution output by the maxSAT solver is, the maxSAT solver will in any case always output a solution that meets all hard constraints. Therefore our model will always output a timetable that obeys all strictly required constraints. While that is indeed better than no timetable at all, it is still important to evaluate this solution, especially in comparison to the timetable produced by Van der Knaap [11].

It bears reiteration that Van der Knaap's [11] resulting timetables also obey certain comfort constraints. The fact that they do means that she succeeded in incorporating comfort constraints without overconstraining her model. This in large part due to the fact that she simply assumed that enforcing particular frequency constraints on separate train series would not cause any conflicts, and running her solver indeed proved that this was true. It is however not always obvious that any number of comfort constraints can be added, as is demonstrated in the problem description in Chapter 1.4. Therefore even though Van der Knaap's model does find a feasible timetable for the instance used, the type of comfort constraints she uses may well cause conflicts on bigger instances. If we want to see how well maxSAT_s does when it is not known on forehand if certain comfort constraints are able to be met, we should compare maxSAT_s to Van der Knaap's model that does not include comfort constraints.

Because we found that her model *is* able to produce a timetable including many comfort constraints, and because maybe larger instances will also allow for the possibility of including said comfort constraints, we will compare our model to both versions of Van der Knaap's model. We refer to her original model as RvdK1 and to the version without any comfort constraints as RvdK2.

4 Results

As mentioned in Section 3.3.1, our model maxSAT_s employs a fixed order within all train series. We first show that this indeed speeds up the solving process by comparing it to the case in which no orders are fixed within any trains in Section 4.1. Then we show the results of maxSAT_s and compare them to those of the models RvdK1 and RvdK2, before exploring the use of fixing the orders within trains of different train series in Section 4.2.

4.1. Results from fixing orders within train series

We compare for the several instances the objective values found with and without fixed orders within train series. In Section 4.2 we discuss the four instances in detail, but for this section it is mostly important to know that each instance has some number of conflicting frequency constraints, where one train series has frequency 3 and another frequency 2 and both PF and SF constraints are enforced. Furthermore, the instances become increasingly more 'difficult'; Instance 1 is particularly small and contains one pair of conflicting train series, Instance 2, 3, and 4 all operate on a much larger network and contain respectively 1, 2, and 3 pairs of conflicting train series.

We call the control sample (no fixed orders at all) maxSAT_x , and as mentioned before, we call the model with fixed orders within train series as described in Section 3.3.1 previously maxSAT_s.

Table 4.1 shows the objective values found by maxSAT_s and maxSAT_x. For the 'simpler' instances 1 and 2, maxSAT_s and maxSAT_x often find the same values, even though for instances 1(c) and 2(c) maxSAT_x is not able to prove within an hour that the found value is optimal. Additionally, maxSAT_s performs better for instances 1(d) and 2(d). Instances 3 and 4 show a similar result, except that for instance 3(d), maxSAT_s reports a worse objective than maxSAT_x. Running this instance 5 times repeatedly gave the same result.

	Objective value															
	1(a)	1(b)	1(c)	1(d)	2(a)	2(b)	2(c)	2(d)	3(a)	3(b)	3(c)	3(d)	4(a)	4(b)	4(c)	4(d)
maxSAT_s	3*	3*	5*	20*	3*	3*	5*	20*	6	6	10	130[a]	7	7	32	34.0[b]
maxSAT_x	3*	3*	5	24	3*	3*	5	34	12	6	103	122 [c]	15	41	39	84.0[d]

[a]Average of 5 runs
[b]See *a*
[c]See *a*
[d]See *a*

Table 4.1: Comparison of the objective values reached by maxSAT_s and maxSAT_x for all instances. Objective values that were proved optimal by the maxSAT solver are marked.

Next to the objective value, it is important to compare how long the different models take to reach an equal objective value. Since most of the time maxSAT_s reports a lower objective value than maxSAT_x, we report the time needed to find maxSAT_x's objective in Figure 4.1.

Not surprisingly, maxSAT_s finds maxSAT_x's objective faster in most instances. There are however some exceptions and side notes. In instance 1(a),(b) and (c), maxSAT_s and maxSAT_x take almost the same time to reach the optimal objective value. This can in part be explained by the small size of Instance 1: since it is already quite small, solving it does not take too long even with the extra possibilities of shuffling the order of trains within a train series. Remember however that maxSAT_x is not able to prove optimality for the objectives found in instance 1(c) and (d).

Furthermore, Figure 4.1b shows that for Instance 2(d), maxSAT_x finds its objective 34 (reported in Table 4.1) faster than maxSAT_s. However, maxSAT_s 'catches up' later and proceeds to find a better objective, for which it is able to prove optimality.

In Figure 4.1c, maxSAT_x finds for Instance 3(a) its objective 12 (Table 4.1) almost at the same time as maxSAT_s, although the latter again finds a better objective within an hour than the former. Furthermore, maxSAT_x unexplicably outperforms maxSAT_s both in time and in objective value in Instance 3(d), though on both counts not by that much.

Figure 4.1d shows that for Instance 4 the two models behave as expected.

These results convince us that it helps the maxSAT solver to fix the order within train series. We therefore continue with maxSAT_s in Section 4.2.

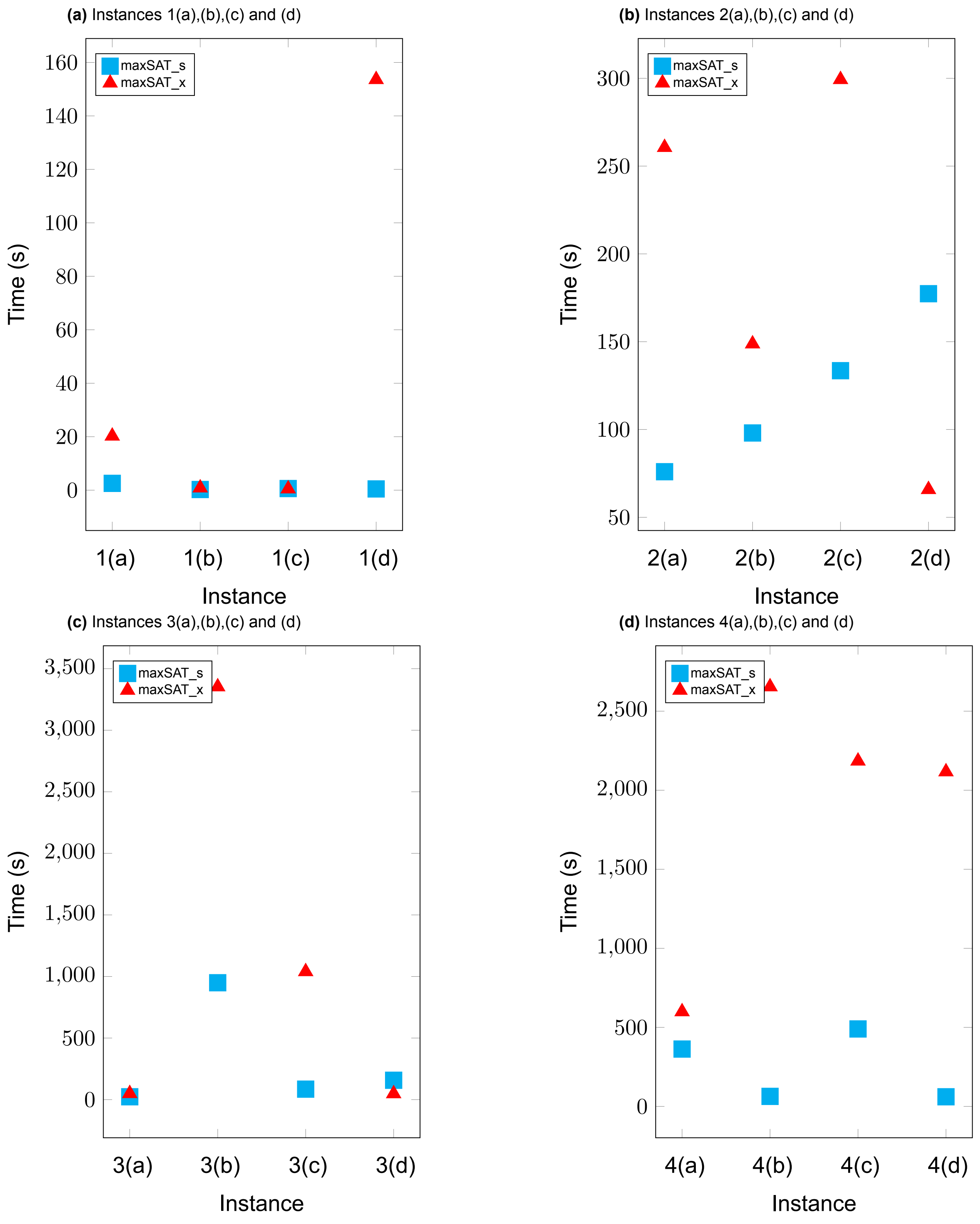


Figure 4.1: Comparison of time needed until maxSAT_x and maxSAT_s reached the objective value found by maxSAT_x for instances 1(a),(b),(c),(d), 2(a),(b),(c),(d), 3(a),(b),(c),(d) and 4(a),(b),(c),(d).

4.2. Instance descriptions and their results

We test our model, which we will refer to from now on as maxSAT_s, on several instances to check its viability. Each instance will have some soft frequency constraints imposed, while all instances will be forced to obey the hard safety constraints. As mentioned before we will ask the model to adhere to different soft constraints, and we distinguish frequency constraints on the train series itself (labeled PF for particular frequency) from frequency constraints on trains of different train series (labeled SF for shared frequency). There are 4 instances which will be described below, in addition to each instance having 4 subinstances that employ different weights:

a All weights equal 1. This is equivalent to the maxSAT solver trying to satisfy as many clauses as possible, and it does not favour any type of frequency constraint;

b Overpowered weights for the shared frequency constraints. The shared frequency constraints get such a high weight that violating even one of the subconstraints of that constraint costs more than violating all of the subconstraints of a particular frequency constraint;

c Overpowered weights for the particular frequency constraints. This instance is like the instance above but for particular frequency constraints;

d ‘Close’ weights for both types of constraints. In this instance, violating all particular frequency constraints costs exactly as much as violating all shared frequency constraints.

This means that all constraints of the same type (PF constraints and SF constraints) will always have the same weights. Each time we will compare the outcome of our model to that of Van der Knaap’s with two different setups: the first is Van der Knaap’s model as it was handed to us: having only particular frequency constraints imposed (RvdK1), and the second is Van der Knaap’s model with no frequency constraints imposed at all (RvdK2). Note that although these models do not use a maxSAT solver, we can still calculate their objective by checking which soft constraints the models do not satisfy and adding up their according weights.

Next to that, we will test the hypothesis that enforcing a certain order to all trains within a group will cut away duplicate solutions such that the maxSAT solver will find better or equally good solutions faster. We call this model maxSAT_o and we will compare the results to the solution of maxSAT_s without a preset order. We employ the three different orders discussed in Section 3.3.1, where for order 3 we use that within the train groups in our experiments, there are always two train series t and t' of which one has frequency 3 and the other frequency 2.

1. The order inside the timetable created by RvdK1;
2. The order inside the timetable created by maxSAT_s after running the maxSAT solver for 1 hour.
3. The following alternating order:

$$d(t_1) < d(t_1') < d(t_2) < d(t_2') < d(t_3).$$

The experiments were run on a laptop with an Intel(R) Core(TM) i5-6200U CPU @2,40 GHz Processor with 7,9 GB RAM memory. Every time the maxSAT solver is called, a timeout is set for 1 hour. We will now look into the different instances.

4.2.1. Instance 1: Single frequency conflict on a small railway network

The first instance is very small and its network is only occupied by train series 1 and 2. Trains from train series 1 travel from station A to B to C and trains from series 2 from station A to B to

D. Therefore they both travel the same route between stations A and B. Note that both trains only move in one direction in this instance. Train series 1 departs three times per hour, while train series 2 only departs twice an hour. Both train series are asked to obey frequency constraints on their own trajectory (particular frequency constraints) *and* together (shared frequency constraints). That is, train series 1 should have a train leaving roughly every 20 minutes and train series 2 should depart roughly every half hour, in addition to the wish of the total of the five trains together leaving roughly every 12 minutes. Figure 4.2 displays this instance, and the resulting timetables are shown in Table 4.2.

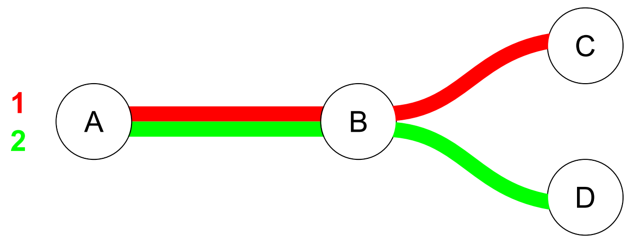


Figure 4.2: Instance 1. Train series 1 (red) departs three times an hour train series 2 (green) two times an hour.

1a		1b		1c		1d	
departure from A		*departure from A*		*departure from A*		*departure from A*	
series	dep. minute	series	dep. minute	series	dep. minute	series	dep. minute
1	:03	2	:13	1	:01	1	:01
1	:15	2	:25	2	:07	2	:07
1	:25	1	:37	1	:21	1	:21
2	:37	1	:48	2	:35	2	:35
2	:50	1	:59	1	:43	1	:43

(a) Timetables for instances 1(a),(b),(c) and (d) from the conflicting train series (1 and 2) produced by maxSAT_s. Departure times of trains belonging to the same series are marked in the same colour.

	1(a) all weights equal 1		1(b) SF weight 5, PF weight 1		1(c) SF weight 1, PF weight 11		1(d) SF weight 4, PF weight 10	
model	obj. value	# violated clauses	obj. value	# violated clauses	obj. value	# violated clauses	obj. value	# violated clauses
maxSAT_s	3	3	3	3	5	5	20	5
RvdK1	7	7	35	7	7	7	28	7
RvdK2	12	12	44	12	52	12	72	12

(b) Found objective values and no. of violated clauses of maxSAT_s compared to RvdK1 and RvdK2 for instances 1(a),(b),(c) and (d)

Instance	Time until obj. reached	Time until optimality proved
1(a)	2.5 s	4.3 s
1(b)	0.2 s	4.4 s
1(c)	0.6 s	224.1 s
1(d)	412.5 s	663.0 s

(c) Time spent in maxSAT solver by maxSAT_s until found objective is reached and proved optimal for instances 1(a),(b),(c) and (d)

Table 4.2: Results for instances 1(a),(b),(c) and (d) from maxSAT_s with maxSAT solver timeout of 1 hour

In this very small instance, it seems like the weights are doing their job. Instance 1(b) favours the shared frequency (SF) constraints: only particular frequency constraints are violated and therefore the five trains together leave in beautiful more or less 12-minute intervals: 12-12-11-11-14. Instance 1(c) favours particular frecuency (PF) constraints, and therefore the intervals of the train departures are fixed to satisfy the particular frequencies. Train series 1 leaves at intervals of 20-22-18, while train series 2 leaves at the intervals of 28 and 32.

In instance 1(a) it is proved that if all constraints are worth the same, it is most economical to violate 3 PF constraints. This hatches in the following way: Train series 1 departs with the intervals 12-10-38, train series 2 with the intervals 13 and 47, and the train series together leave in intervals of 12-10-12-13-13. Train series 2 only had one PF constraint:

$$d(t_A^{2,1}) - d(t_A^{2,2}) \in [28, 32]_{60},$$

and it clearly got violated. Two of train series 1's PF constraints got violated,

$$d(t_A^{1,1}) - d(t_A^{1,2}) \in [18, 22]_{60} \cup [38, 42]_{60}$$

and

$$d(t_A^{1,2}) - d(t_A^{1,3}) \in [18, 22]_{60} \cup [38, 42]_{60}$$

but the following constraint did not get violated:

$$d(t_A^{1,1}) - d(t_A^{1,3}) \in [18, 22]_{60} \cup [38, 42]_{60}$$

as $d(t_A^{1,1}) - d(t_A^{1,3}) = 3 - 25 = -22 \in [-42, -38] \cup [-22, -18]$. This is interesting as the maxSAT solver indicates that it only violated 2 out of 3 PF constraints of train series 1, while neither interval – not 12, 10 or 38 – is close to 20.

In instance 1(d), five SF clauses are violated, exactly like in instance 1(c). It could be a coincidence that the solver always finds this optimal value, but our experience has shown that this happens every run. This could mean that the weights are not as balanced as we hypothesized, because apparently it is not possible to violate 4 PF constraints and 0 SF constraints.

As for the maxSAT_o, equipped with any order between trains in train series with conflicting F-constraints it finds all the same objectives as maxSAT_s. As can be seen in Table 4.3, maxSAT_o also succeeds in proving optimality for all objectives, just like maxSAT_s did. Furthermore, it even proves optimality in almost exactly the same time frames (see Figure 4.3).

	Objective value			
model	1(a)	1(b)	1(c)	1(d)
maxSAT_s	3*	3*	5*	20*
maxSAT_o: RvdK1 order	3*	3*	5*	20*
maxSAT_o: alternating order	3*	3*	5*	20*
maxSAT_o: maxSAT order	3*	3*	5*	20*

Table 4.3: Objective values obtained by maxSAT_s and maxSAT_o equipped with different orders for instances 1(a),(b),(c) and (d). Objectives that were proved optimal by the maxSAT solver are marked. (all of them, in this case)

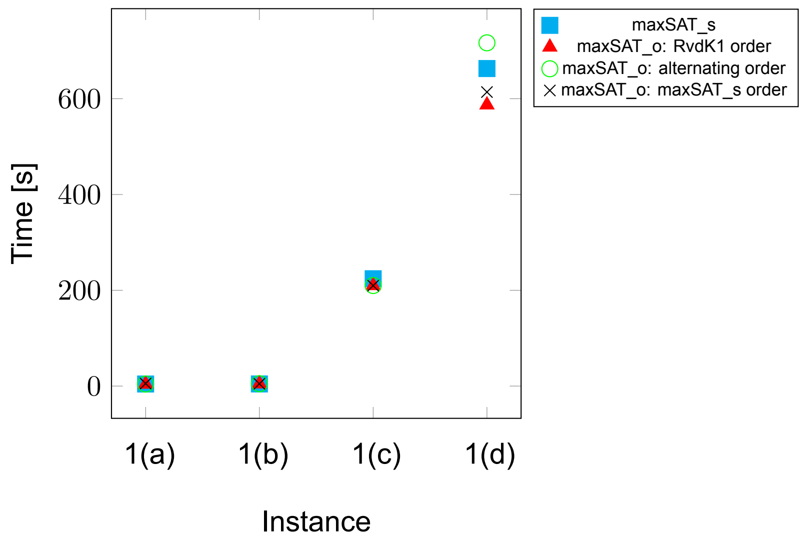


Figure 4.3: Time until optimality proved for found objective values by maxSAT_s and maxSAT_o equipped with with different orders for instances 1(a),(b),(c) and (d)

4.2.2. Instances 2,3 and 4

The following instances include a much larger and more realistic network: the Dutch railway network in the top of Noord-Holland. This network is also used by Van der Knaap [11] and Vollebergh [20] to test their models and Figure 4.4 shows this network.

Its lineplanning contains a total of twelve train series, which can be viewed in table 4.4. See Appendix C for the station names. The train network is occupied by 6 'Intercitys' (inter city trains) and 6 'Sprinters' (regional trains). Every pair of two train series that have consecutive indices are in reality referred to as a single train series, but for clarity we refer to these as two separate train series, one moving forwards on the route and the other moving backwards. Most train series have a frequency of two departures per hour, but train series 1 and 2 and train series 5 and 6 both depart three times every hour. What follows next is an instance-by-instance account of the timetables resulting from maxSAT_s, objective values and a description of the optimality attained by the maxSAT solver, starting with Instance 2. This is followed by an evaluation of the model maxSAT_o equipped with the discussed orders.

Index	Train type	First station	Final station	Stops	Frequency
1	IC	Obpa	Amr	Ass, Zd, Cas, Amr	3
2	IC	Amr	Obpa	Amr, Cas, Zd, Ass	3
3	IC	Obpa	Hdr	Ass, Zd, Cas, Hlo, Amr, Amrn, Hwd, Sgn, Ana, Hdrz, Hdr	2
4	IC	Hdr	Obpa	Hdr, Hdrz, Ana, Sgn, Hwd, Amrn, Amr, Hlo, Cas, Zd, Ass	2
5	IC	Obpa	Ekz	Ass, Hn, Hnk, Bkg, Bkf, Ekz	3
6	IC	Ekz	Obpa	Ekz, Bkf, Bkg, Hnk, Hn, Ass	3
7	S	Obpa	Utg	Ass, Zd, Kz, Zzs, Wm, Kma, Utg	2
8	S	Utg	Obpa	Utg, Kma, Wm, Zzs, Kz, Zd, Ass	2
9	S	Obpa	Hn	Ass, Hw, Hlms, Hlm, Bll, Sptz, Sptn, Drh, Bv, Hk, Utg, Cas, Hlo, Amr, Amrn, Hwd, Obd, Hn	2
10	S	Hn	Obpa	Hn, Obd, Hwd, Amrn, Amr, Hlo, Cas, Utg, Hk, Bv, Drh, Sptn, Sptz, Bll, Hlm, Hlms, Hw, Ass	2
11	S	Aeg	Hnk	Ass, Zd, Zdk, Pmw, Pmr, Pmo, Hn, Hnk	2
12	S	Hnk	Aeg	Hnk, Hn, Pmo, Pmr, Pmw, Zdk, Zd, Ass	2

Table 4.4: Line planning in instances 2, 3 and 4. See Appendix C for an explanation of the station names.

Instance 2: Single frequency conflict on the railway network of the top of Noord-Holland
We impose a conflict on two of the twelve train series. Train series 1 and 3 share part of their route, from Opba to Amr, and the line planning shows that train series 1 is set to depart 3 times an hour and train series 2 twice an hour. Again we impose separate frequency constraints on the respective train series (PF constraints) and a frequency constraint for the five trains grouped together (SF constraints). All other train series depart two times an hour and have soft PF constraints imposed on them. The results in Table 4.5c below are divided into the different weight instances (a),(b),(c) and (d) explained at the start of Section 4.2.

2a	
departure from Obpa	
series	dep. minute
1	:08
1	:22
3	:34
1	:44
3	:58

2b	
departure from Obpa	
series	dep. minute
3	:03
1	:13
1	:27
3	:37
1	:51

2c	
departure from Obpa	
series	dep. minute
1	:05
3	:11
1	:25
3	:39
1	:47

2d	
departure from Obpa	
series	dep. minute
1	:00
3	:06
1	:20
3	:34
1	:42

(a) Timetables for instances 2(a),(b),(c) and (d) from the conflicting train series (1 and 2), produced by maxSAT_s. Departure times of trains belonging to the same series are marked in the same colour.

	2(a) all weights equal 1		**2(b) SF weight 5, PF weight 1**		**2(c) SF weight 1, PF weight 11**		**2(d) SF weight 4, PF weight 10**	
model	obj. value	# violated clauses	obj. value	# violated clauses	obj. value	# violated clauses	obj. value	# violated clauses
maxSAT_s	3	3	3	3	5	5	20	5
RvdK1	6	6	30	6	6	6	24	6
RvdK2	22	22	50	22	182	22	172	22

(b) Found objective values and no. of violated clauses of maxSAT_s compared to RvdK1 and RvdK2 for instances 2(a),(b),(c) and (d)

Instance	**Time until obj. reached**	**Time until optimality proved**
2(a)	76.0 s	80.0 s
2(b)	98.0 s	99.4 s
2(c)	133.5 s	173.1 s
2(d)	1400.2 s	1832.7 s

(c) Time spent in maxSAT solver by maxSAT_s until found objective is reached and proved optimal for instances 2(a),(b),(c) and (d)

Table 4.5: Results for instances 2(a),(b),(c) and (d) from maxSAT_s with maxSAT solver timeout of 1 hour

As can be seen in Table 4.5b, for instance 2(a) it is possible to make a timetable that only violates 3 clauses, which results in a timetable in which the intervals of train series 1 are not very good (two PF subconstraints are violated, and hence two intervals are 'bad'; Two of the three intervals do not fall inside the permitted $[18, 22] \cup [38, 43]$-borders: Table 4.5a shows that the intervals are 14-22-24, so only 22 is the right number of minutes. The intervals of train series 3 seem better because maxSAT_s finds that only one PF constraint is violated, but of course there *is* only one pair of departures within train series 3, so this results in intervals of 24 and 36 $\notin [28, 32]$. All SF constraints are honored, which means that the five trains together leave every 10-14 minutes, and they do: 14-12-10-14-10.

In instance 2(b), where the shared frequency constraints are favoured such that violating only one of them is worse than violating all PF constraints, again all SF constraints are sustained with intervals that look like 10-14-10-14-12. 3 PF constraints are violated, just like in instance (a), causing intervals such as 34-26 and 14-24-22.

Instance 2(c) is instructed to favour particular frequency constraints and it shows: train series 1 leaves every 20-22-18 minutes, train series 3 every 28-32 minutes, but together they leave in the fickle intervals 6-14-14-8-18, violating 5 subconstraints (out of a total of 10).

Instance 2(d), which is instructed not to favour any type of constraint over the other, finds that the best objective value again causes 5 SF clauses to be violated, making the intervals of the two train series together 6-14-14-8-18, while the other train series behave well in terms of intervals:

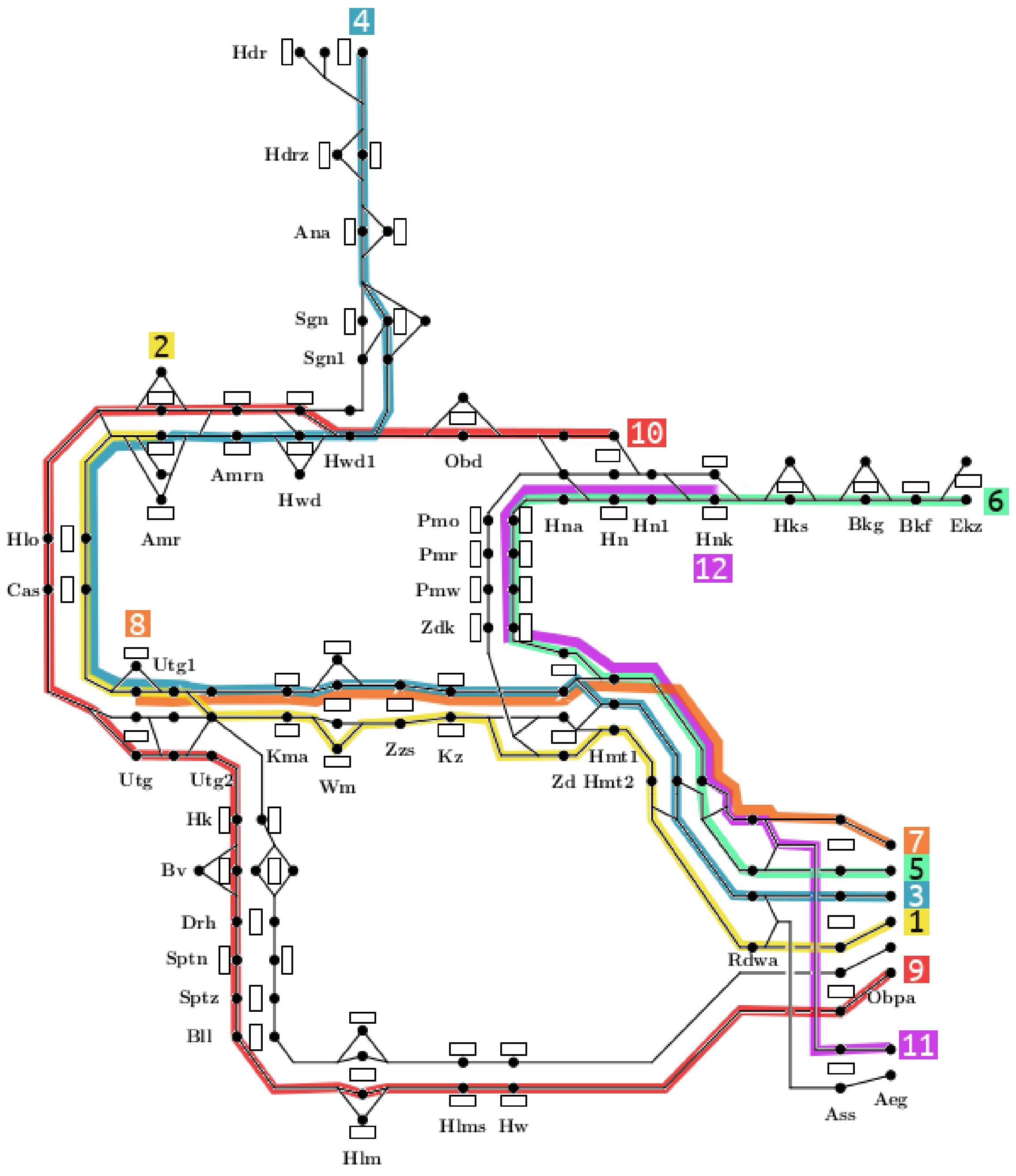


Figure 4.4: The train network in the top of Noord-Holland. Adapted from Van der Knaap [11]. The coloured routes indicate the stage points that the train series follow.

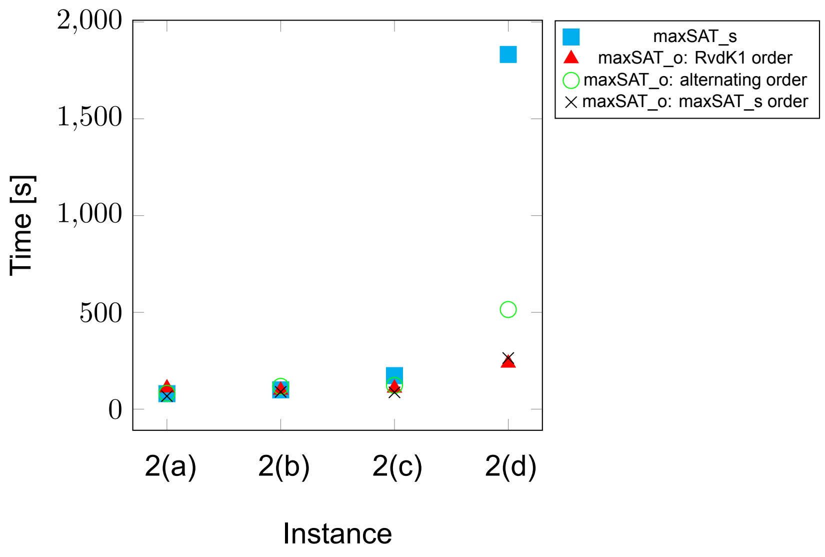


Figure 4.5: Time until optimality proved for found objective value by maxSAT_s with different imposed orders for Instance 2(a),(b),(c) and (d)

28-30 and 20-22-18.

Next to the very small Instance 1, Instance 2 is the only instance for which the maxSAT solver finds optimal values within an hour. Not surprisingly, the optimal objective values found by the solver are lower than the objective values resulting from the timetables of RvdK1 and RvdK2. Therefore for this instance maxSAT_s performs better than the old models.

	Objective value			
model	2(a)	2(b)	2(c)	2(d)
maxSAT_s	3*	3*	5*	20*
maxSAT_o: RvdK1 order	3*	3*	5*	20*
maxSAT_o: alternating order	3*	3*	5*	20*
maxSAT_o: maxSAT order	3*	3*	5*	20*

Table 4.6: Objective values obtained by maxSAT_s and maxSAT_o equipped with different orders for instances 2(a),(b),(c) and (d). Objectives that were proved optimal within the hour are marked (in this case again all of them)

Since Instance 2 is quite small conflict-wise, maxSAT_s proves that it can find all optimal solutions and, not surprisingly, so does maxSAT_o. Therefore instead of showing the objective values they find, we show the time in which it finds the optimal values in Figure 4.5. For instance 2(a) and 2(b), there is barely any difference between the time in which the models find the optimal solution. Instance 2(c) does show that any fixed order does help find a solution a bit faster, but the difference is especially large for instance 2(d). Here, maxSAT_o equipped with an alternating order is 4 times faster than maxSAT_s, and equipped with the order found by maxSAT_s itself, maxSAT_o is even 10 times faster.

Instance 3: Two frequency conflicts on the railway network of the top of Noord-Holland
We continue on Instance 2, adding one more conflict. Just like train series 1 and 3, train series 6 and 12 depart respectively two and three times an hour. Next to the separate soft frequency constraints that are already in place we impose a joint soft frequency constraint on the train series from the stage point where they first share part of their route, at Hnk, and we observe the results in Table 4.7.

3a		3b		3c		3d	
departure from Obpa		*departure from Obpa*		*departure from Obpa*		*departure from Obpa*	
series	dep. minute	series	dep. minute	series	dep. minute	series	dep. minute
1	:04	3	:05	3	:13	3	:15
3	:15	1	:15	1	:18	1	:28
1	:29	1	:29	1	:36	1	:37
3	:41	3	:39	3	:43	3	:45
1	:51	1	:53	1	:56	1	:51
departure from Hnk		*departure from Hnk*		*departure from Hnk*		*departure from Hnk*	
series	dep. minute	series	dep. minute	series	dep. minute	series	dep. minute
6	:00	6	:06	6	:17	12	:01
12	:12	6	:34	12	:28	6	:27
6	:27	12	:39	6	:36	12	:32
12	:35	12	:49	6	:55	6	:37
6	:41	6	:52	12	:56	6	:59

(a) Timetables for instances 3(a),(b),(c) and (d) from the conflicting train series (1 and 2, and 3 and 6) produced by maxSAT_s. Departure times of trains belonging to the same series are marked in the same colour.

	3(a) all weights equal 1		**3(b) SF weight 5, PF weight 1**		**3(c) SF weight 1, PF weight 11**		**3(d) SF weight 4, PF weight 10**	
model	obj. value	# violated clauses	obj. value	# violated clauses	obj. value	# violated clauses	obj. value	# violated clauses
maxSAT_s	6	6	6	6	10	10	130	19
RvdK1	12	12	60	12	12	12	48	12
RvdK2	27	27	75	27	177	27	198	27

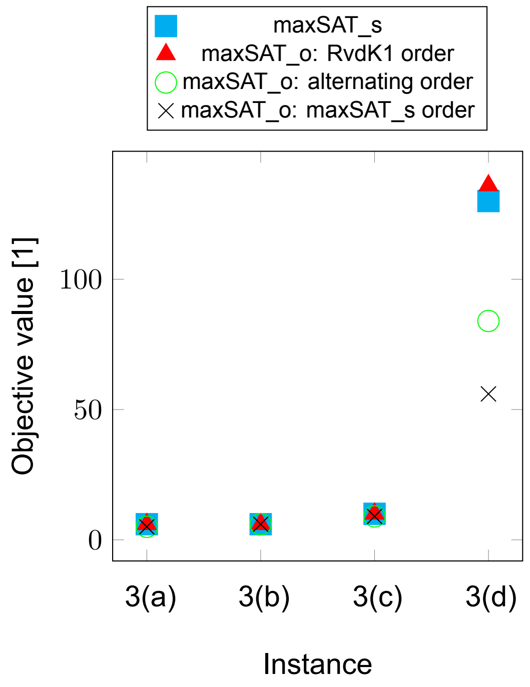
(b) Found objective values and no. of violated clauses of maxSAT_s compared to RvdK1 and RvdK2 for instances 3(a),(b),(c) and (d)

Instance	Time until obj. reached	Time until optimality proved
3(a)	593.1 s	>3600 s
3(b)	496.8 s	>3600 s
3(c)	2093.8 s	>3600 s
3(d)	3247.1 s	>3600 s

(c) Time spent in maxSAT solver by maxSAT_s until found objective is reached and proved optimal for instances 3(a),(b),(c) and (d)

Table 4.7: Results for instances 3(a),(b),(c) and (d) from maxSAT_s with maxSAT solver timeout of 1 hour

For Instance 3, Table 4.7c shows that the maxSAT solver is finding it harder to solve the problem as it is not able to confirm that it has solved the problem to optimality within an hour for any of the instances. Since the added conflict is very similar to the single conflict in Instance 2 and the conflicts do not interfere with each other, it could be that for instances 3(a)-(c), optimality is in fact attained, as the objective values are double the objective values in instances 2(a)-(c). This hypothesis is supported by the fact that the subconstraints the solver failed to satisfy are six PF clauses, of which two are from train series 1 and one is from train series 3, and another two from



(a) Illustration of the differences in found objective values

	Objective value			
model	3(a)	3(b)	3(c)	3(d)
maxSAT_s	6	6	10	130
maxSAT_o: RvdK1 order	6	6	10	136
maxSAT_o: alternating order	5*	6	9	84
maxSAT_o: maxSAT order	5*	6*	9	56

(b) Exact values of the objectives. Objectives that were proved optimal within the hour are marked.

Figure 4.6: Objective values found by maxSAT_s and maxSAT_o equipped with with different orders for instances 3(a),(b),(c) and (d)

train series 6 and the last one from train series 12, as if the two pairs of train series are exactly alike. Therefore the maxSAT solver probably just did not have enough time to rule out all other solutions.

For instance 3(d), the model RvdK1 actually has a lower objective value than maxSAT_s. This means that the solution attained by one hour of running the maxSAT solver is worse than a solution found by opting not to incorporate the shared frequency constraints at all.

Moving on to the tests with additional imposed order constraints, the results of which can be viewed in Figure 4.6, it seems like maxSAT_o does consistently equally well or better when handed the order that was found by maxSAT_s after an hour of running the maxSAT solver. In most cases, maxSAT_o does equally well with order RvdK1 as maxSAT_s did, though not in instance 3(d). Even a 'random' order helps maxSAT_o quite well. However, if the solver performs equally well with an added imposed order, adding these extra constraints does not help the solving process much. In this case, it *is* noteworthy that while maxSAT_s fails to prove optimality for any of the instances 3(a)-(d), maxSAT_o aided by the alternating order *is* able to do so in case 3(a) and 3(b). For instance 3(a) it even finds that the optimal value is not 6 as we hypothesized, but 5.

This is an interesting phenonemon. We mentioned before that it was likely that 6 was the optimal objective for 3(a), yet the solver *proves* in maxSAT_o with alternating order that the optimal objective is 5. If we take a look at the clauses that were violated we find that they belong to the following violated subconstraints:

3a	
maxSAT_s	**maxSAT_o with alternating order**
PF subconstr. of series 1	PF subconstr. of series 1
PF subconstr. of series 1	PF subconstr. of series 1
PF subconstr. of series 3	PF subconstr.of series 3
PF subconstr. of series 6	SF subconstr. between series 6 and 12
PF subconstr. of series 6	PF subconstr. of series 12
PF subconstr. of series 12	

Table 4.8: Violated subconstraints in instance 3(a) found by maxSAT_s and maxSAT_o equipped with alternating order. The top half of the table concerns subconstraints of the conflict within train series 1 and 3, while the lower half concerns the conflict within train series 6 and 12.

We note that on the route part that train series 3 and 6 share, it is apparently possible to violate one SF subconstraint and one PF subconstraint instead of three PF subconstraints. This means that all intervals should be close to perfect except one for both the PF constraints and the SF constraints. If we look at the resulting timetables in Figure 4.9, we see in Figure 4.9a that indeed all SF subconstraints except one are satisfied: only between the first train of series 6 (departing at :05) and the second train of series 6 (departing at :31), an interval of 26 is found which is not within the limits of $[10, 14]_{60} \cup [22, 26]_{60} \cup [34, 38]_{60} \cup [46, 50]_{60}$. However, more than just one PF subconstraint of train series seems to get violated: the intervals between the trains are 20, 26 and 14, so two of the three are not inside $[18, 22]_{60} \cup [38, 42]_{60}$. The reason for this discrepancy is that PF constraints only get enforced on the starting point of any train series to reduce the number of clauses, and Hnk is not the starting point of train series 6 but the stage point where series 6 and 12 first travel the same route, as can be seen in Figure 4.4. It is also not necessary to enforce regular frequencies on all stage points that a train series follows: because all trains of the same type have the same driving time for any particular stage, the only form of slack is that in driving and stopping. The size of the slacks will be randomly adjusted to be larger or smaller (although never larger than `maxSlackTime` and `maxStopTime`) in order to make all train trips fit in the timetable. Therefore the slacks will probably not heap up to the point where all trains in a train series leave during the first fifteen minutes.

We can indeed see in Figure 4.9b that train series 6 leaves from Ekz in near-perfect intervals of 20-18-22. Hence the maxSAT solver did not violate the PF constraints of train series, and it was able to find the optimal solution 5.

3a	
departure from Hnk	
series	dep. minute
6	:05
12	:17
6	:31
12	:41
6	:51

(a) Timetable for train series 6 and 12 departing from Ekz

3a	
departure from Ekz	
series	dep. minute
6	:06
6	:26
6	:44

(b) Timetable for train series 6 departing from Ekz

Table 4.9: Timetables for instance 3a resulting from maxSAT_o equipped with alternating order within conflicting time series.

Instance 4: Three frequency conflicts on the railway network of the top of Noord-Holland
This instance has the same line planning as instance 3, but in addition we ask that trains from train series 6 and 10 leave in a regular frequency. Therefore we ask three train series in total to depart at even intervals: train series 1 and 3, 6 and 12 and 6 and 10.

For instance 4 it is interesting that although a third equally large conflict is added to the problem, at least for instances (a) and (b) the objective found is not 1.5 times as large, as can be seen in Table 4.10b. This is most likely due to the fact that the conflict between train series 6 and 10 also plays into the conflict of train series 6 and 12 in a positive manner: the PF constraints of train series 6 only need to be breached once for both conflicts to be alleviated somewhat. In fact, the constraints violated in Instance 3(a) were two PF constraints of train series 1, one PF constraint of train series 3, two PF constraints of train series 6, and one PF constraint of train series 12. In Instance 4(a) the exact same types of constraints were violated in addition to a PF constraint of train series 10.

Furthermore, the maxSAT solver 'loses' from the model RvdK1 in instance (c) already. It seems like the wall that is the time shortage seems to be moving up as the instances are getting harder. Note that RvdK1 does not violate all shared frequency constraints, of which there are a total of $\binom{5}{2} + \binom{5}{2} + \binom{5}{2} = 30$. Some frequency constraints get adhered to 'by chance'.

As far as the results for maxSAT_o go, we see in Figure 4.6a that it seems to help less to fix the orders than it did for Instance 3. Interestingly, maxSAT_o with alternating order does much better than the other models for instances 4(c) and (d), but worse for instances 4(a) and 4(b).

Imposing the RvdK1 order causes the objective value to be worse than the value found in maxSAT_s for Instance 4(c), but for the other instances it is equal or better.

The model maxSAT_o aided by the seemingly most promising order, found by maxSAT_s with a solver timeout of an hour, performs varyingly. The objective maxSAT_o finds is worse than that of maxSAT_s in instance 4(a) and equal in instance (b). Yet in Instance 4(b) it is able to confirm that 7 is the optimal value, where the other models are not. This means that the imposed order has decreased the solving time so much that the solver was able to finish optimizizing. It again performs much worse that maxSAT_s for instance (c), and then better for instance (d).

4a		4b		4c			4d	
departure from Obpa		*departure from Obpa*		*departure from Obpa*			*departure from Obpa*	
series	dep. minute	series	dep. minute	series	dep. minute		series	dep. minute
1	:06	1	:04	3	:05		1	:07
3	:18	1	:16	1	:09		3	:20
3	:32	3	:28	1	:29		1	:26
1	:44	3	:42	3	:33		1	:36
1	:54	1	:54	1	:51		3	:48
departure from Hnk		*departure from Hnk*		*departure from Hnk*		-	*departure from Hnk*	
series	dep. minute	series	dep. minute	series	dep. minute		series	dep. minute
12	:06	6	:07	6	:13		12	:21
6	:16	6	:18	12	:23		6	:25
12	:29	12	:20	6	:34		6	:41
6	:30	6	:49	6	:50		12	:53
6	:58	12	:58	6	:51		6	:55
departure from Hn		*departure from Hn*		*departure from Hn*			*departure from Hn*	
series	dep. minute	series	dep. minute	series	dep. minute		series	dep. minute
10	:09	10	:05	6	:13		10	:18
6	:16	6	:07	10	:31		6	:25
6	:30	6	:18	6	:34		10	:33
10	:35	10	:27	6	:50		6	:41
6	:58	6	:49	10	:59		6	:55

(a) Timetables for instances 4(a),(b),(c) and (d) from the conflicting train series (1 and 2, 3 and 6 and 3 and 5) produced by maxSAT_s. Departure times of trains belonging to the same series are marked in the same colour.

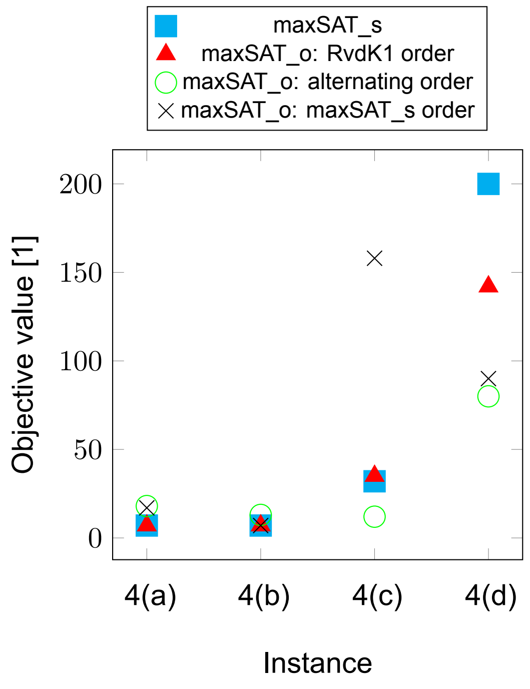
	4(a) all weights equal 1		**4(b) SF weight 5, PF weight 1**		**4(c) SF weight 1, PF weight 11**		**4(d) SF weight 4, PF weight 10**	
model	obj. value	# violated clauses	obj. value	# violated clauses	obj. value	# violated clauses	obj. value	# violated clauses
maxSAT_s	7	7	7	7	32	12	200	32
RvdK1	20	20	100	20	20	20	80	20
RvdK2	32	32	100	32	182	32	218	32

(b) Found objective values and no. of violated clauses of maxSAT_s compared to RvdK1 and RvdK2 for instances 4(a),(b),(c) and (d)

Instance	Time until obj. reached	Time until optimality proved
4(a)	2824.3 s	>3600 s
4(b)	3354.6 s	>3600 s
4(c)	3532.5 s	>3600 s
4(d)	3376.9 s	>3600 s

(c) Time spent in maxSAT solver by maxSAT_s until found objective is reached and proved optimal for instances 4(a),(b),(c) and (d)

Table 4.10: Results for instances 4(a),(b),(c) and (d) with maxSAT solver timeout of 1 hour



(a) Illustration of the differences in found objective values

model	Objective value 4(a)	4(b)	4(c)	4(d)
maxSAT_s	7	7	32	200
maxSAT_o: RvdK1 order	7	7	35	142
maxSAT_o: alternating order	18	13	12	80
maxSAT_o: maxSAT_s order	17	7*	158	90

(b) Exact values of the objectives. Objectives proved to be optimal within an hour are marked with an asterisk.

Figure 4.7: Objective values found by maxSAT_s and maxSAT_o equipped with with different orders for instances 4(a),(b),(c) and (d)

5 Discussion & Conclusion

5.1. Conclusion

We have found that the model of Van der Knaap (RvdK1) can be altered such that it always returns a timetable meeting at least all trip-defining constraints and safety constraints, and part of the comfort constraints. Additionally, the new model maxSAT_s returns the comfort constraints that it fails to satisfy. Especially in cases where maxSAT_s receives a black-and-white notion of which constraints are important and which are not, it is able to accurately convey this wish into a timetable that largely meets the favoured constraint and not the other. This is contrary to regular PESP models that only know hard constraints and are therefore unable to produce a timetable in the first place when comfort constraints cause conflicts.

Next to that, the speed of the maxSAT solver increases through the addition of order constraints within train series, and sometimes this makes it possible to yield better results within an equally long time frame. Setting the order within a set of different train series can be risky as it may cut away good solutions with other orders, but for a smaller instance it does help.

Unfortunately, our model maxSAT_s does not always perform better than the model of Van der Knaap, which obtains a solution by simply not taking into account comfort constraints of which we know beforehand that they will conflict with each other. This is because our model starts from scratch and only uses the knowledge that a solution exists, rather than Van der Knaap's solution. Additionally, maxSAT_s is certainly not faster. The maxSAT solver was set to timeout at 1 hour, and even though a model is allowed to take longer than that to devise a timetable for the coming ten years, it already being worse than RvdK1 with three conflicts means that it reaches its limits too quickly. A real-life instance will contain many more conflicts, as in the ideal situation any route part of the entire train network has trains departing in a regular fashion. This means that on all route parts that have different train series occupying them, frequency constraints need to be in place that may or may not be conflicting with each other. It is therefore not a good indicator for future results that the model already gets into trouble so quickly.

Furthermore, due to the manner in which frequency constraints are enforced in PESP, falsifying one frequency subconstraint is not equivalent to not meeting exactly one interval. Additionally, the model is programmed to optimize the inclusion of constraints, but it is not able to adhere slightly to any constraint, only a full-fletched yes or no. This makes it harder to optimize on our real goal: making the intervals between trains as close to the perfect length as possible.

In conclusion, the maxSAT_s model is not yet ready to tackle real-life instances, but other solvers may perform better. In fact, maxSAT may not even be the right optimization tool for the job. Nevertheless, it *is* able to always generate a feasible timetable without knowing which soft constraints will clash and to steer the solution by incorporating weights. Finally, pinning down an order decreases the solution space and is therefore worth looking into in future research.

5.2. Suggestions for improvement of the model

Due to the way that frequency constraints are defined in a PESP where the order of trains is not fixed, our understanding of violating a subconstraint and the weight we assign to it getting disobeyed does not entirely match the effects that breaching such a constraint has.

Furthermore, in some cases the results of the maxSAT solver seem to differ greatly each run. Therefore it could be useful to run all experiments multiple times.

5.3. Suggestions for further research

The maxSAT solver used in this model is quite slow, as it was built to be robust and open-source. It is also quite old as it was built in 2012, so anyone wanting to continue research on the subject with a maxSAT solver would be advised to use a newer solver.

Perhaps it is be better to use a different optimizing technique entirely. The reason we used maxSAT is primarily because NS was already working with a SAT-problem, so it was a logical step towards optimization. However, SAT is very much a feasibility problem and maxSAT optimizes how many entire constraints it can satisfy, not how much it can satisfy a constraint. Solving the problem as an ILP with commercial solvers has the advantage that it is not all-or-nothing when it comes to meeting constraints, which especially in the case of frequency constraints comes close to our goal.

Of course, NS stepped off the ILP solver for a reason, so perhaps a hybrid method may be fruitful. The reason Van der Knaap and Vollebergh's ventures with the SAT-solver are succesful is that they make use of the speed to iterate over the SAT-solver. Therefore for optimizing the adherance to frequency constraints, it could work to optimize a smaller part of the solution first with an ILP solver. A hybrid method could then be to optimize the adherance to all constraints on the part of the network that contains a conflict, and then to remove clashing constraints and to check with a SAT-solver if a solution exists for all remaining constraints on the network.

Right now, the maxSAT-solver views all subconstraints in a binary view: it can either falsify or satisfy them, which means that a time difference between two trains is either perfect or entirely up to the whims of the solver. Having constraints that constitute small steps of deviations from the perfect intervals, with higher weights for more drastic deviations, rids the solver of this problem (although it will also create many more clauses and therefore more possibilities).

As it helps to fix the order between groups of trains, it would be interesting to find out what other factors can be fixed to remove duplicate solutions. If certain fixations decrease running time drastically (even if those fixations cut away optimal solutions), it could be useful to iterate over different fixations to find out which is best.

When the order within groups of trains that have shared frequency constraints imposed on them is fixed, it is no longer necessary to enforce these frequency constraints such as defined in Section 2.2.2. We could single out pairs that follow each other up and enforce a single interval on the time difference between them. Then there would be fewer subconstraints (for n trains in a group, n subconstraints instead of $\binom{n}{2}$) and violating a subconstraint and an interval will become equivalent.

A last suggestion is to incorporate Vollebergh and Van der Knaap's route option mechanism. This would be useful if it is not possible to obey frequency constraints due to overcrowding on the tracks.

References

[1] E.J.W. Abbink. “Crew Management in Passenger Rail Transport”. PhD thesis. Erasmus Universiteit, 2014. url: https://repub.eur.nl/pub/76927.

[2] C. Ansótegui, M.L. Bonet, and J. Levy. “SAT-based MaxSAT algorithms”. In: *Elsevier, Artificial Intelligence 196* (2013), pp. 77–105.

[3] P. Bakker and P. Zwaneveld. *Het belang van openbaar vervoer: De maatschappelijke effecten op een rij*. Tech. rep. Centraal Planbureau en Kennisinstituut voor Mobiliteitsbeleid, 2009, pp. 74–76.

[4] D. Le Berre and A. Parrain. “The Sat4j library, release 2.2”. In: *Journal on Satisfiability, Boolean Modeling and Computation* 7 (2010), pp. 59–64. url: https://content.iospress.com/download/journal-on-satisfiability-boolean-modeling-and-computation/sat190075?id=journal-on-satisfiability-boolean-modeling-and-computation%2Fsat190075.

[5] C. F. Daganzo. *Fundamentals of Transportation and Traffic Operations*. 1st ed. ISBN: 978-0-08-042785-0. Emerald, 2008. url: https://ndl.ethernet.edu.et/handle/123456789/75532.

[6] P. Grossmann. *Polynomial Reduction from PESP to SAT*. Tech. rep. 11-05. Technische Universität Dresden, 2011.

[7] P. Grossmann. “Satisfiability and Optimization in Periodic Traffic Flow Problems”. PhD thesis. TU Dresden, 2016. url: https://nbn-resolving.org/urn:nbn:de:bsz:14-qucosa-213122.

[8] P. Grossmann et al. “Solving Periodic Event Scheduling Problems with SAT”. In: *Advanced Research in Applied Artificial Intelligence*. Vol. 7345. doi:10.1007/978-3-642-31087-4_18. 25th International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems. Springer Berlin Heidelberg, 2012, pp. 166–175.

[9] R. Hoogervorst. *Rolling Stock Scheduling (at NS)*. Part of the guest lecture on Railway Scheduling given during the Mastermath course *Scheduling*. 2021.

[10] D. Huisman. *Operations Research in Passenger Railway Transportation*. Part of the guest lecture on Railway Scheduling given during the Mastermath course *Scheduling*. 2021.

[11] R. Van der Knaap. “Cyclic railway timetabling using an iterative SAT approach with a feedback mechanism”. MA thesis. Tilburg University, 2021. url: https://www.worldcat.org/nl/title/1296574558.

[12] N. Lindner and C. Liebchen. “Timetable merging for the Periodic Event Scheduling Problem”. In: *EURO Journal on Transportation and Logistics* (2022), pp. 77–105.

[13] RTL Nieuws. *Gaan de bekende gele NS-borden nou weg of niet?* 2014. url: https://www.rtlnieuws.nl/nieuws/artikel/2093126/gaan-de-bekende-gele-ns-borden-nou-weg-niet.

[14] *NS in 2021 opnieuw 52 procent minder reizigers dan in 2019*. 2021. url: https://www.spoorpro.nl/materieel/2021/12/28/ns-in-2021-opnieuw-52-procent-minder-reizigers-dan-in-2019/?gdpr=deny.

[15] *NS Jaarverslag in het kort*. 2020. url: https://www.nsjaarverslag.nl/in-het-kort.

[16] G. Polinder. "Resolving infeasibilities in the PESP model of the Dutch railway timetabling problem". MA thesis. TU Delft, 2015. url: http://resolver.tudelft.nl/uuid:ed4c1256-700c-4e04-8da0-d8cb3fa74b8b.

[17] G. Polinder et al. "An iterative heuristic for passenger-centric train timetabling with integrated adaption times". In: *Computers & Operations Research* 142.105740 (2022).

[18] It's Public. *Vergelijking van de CO_2-uitstoot van Verschillende Vervoersvormen*. It's Public is a strategic advisor for the public sector. 2019.

[19] P. Serafini and W. Ukovich. "A mathematical model for periodic scheduling problems". In: *SIAM J. Disc. Math.* 2.4 (1989), pp. 550–581.

[20] M. Vollebergh. "Incorporating flexible track use in the SAT model of the Dutch railway timetabling problem". MA thesis. TU Delft, 2020. url: http://resolver.tudelft.nl/uuid:37aaf584-8775-44d6-b3cb-e914f8a54b48.

A

On total traveler comfort with regard to frequency constraints

In order to optimize a timetable with regard to total traveler comfort, it first needs to be decided what constitutes a ‘good’ timetable as far as traveler comfort goes. It is not hard to agree upon the ideal scenario: this is the (impossible) situation in which on each section of tracks, trains follow a perfectly regular frequency. That is to say, for any pair of adjacent stations A and B, if there are n trains per hour travelling between them, a train should leave station A exactly every $\frac{60}{n}$ minutes. However, it is not evident how to quantify the total traveler comfort in solutions that are less than perfect. This is because obeying a frequency constraint closely at one section may result in the disobedience of a frequency constraint at another. Therefore, in order to define the quality of a solution, we should be able to indicate a quantification of total traveler comfort for any given timetable.

Possible quantifications

One way to quantify traveler comfort is by allotting a value to the time spent waiting, then calculating the total weighted waiting time of all travelers by deducing from an origin-destination matrix at which stations these travelers board the train. In doing so, it can be measured how the total of passengers are affected by low or high adherance to frequency constraints on certain stations.

Bakker, P. et al. [3] write that Dutch institutions such as ProRail, Ecorys and CPB[1] assume that people arrive at stations randomly; meaning that if a train leaves every half an hour, a passenger will on average have to wait for 15 minutes. They note that according to thorough research by Douglas Economics (2006), this assumption holds only when there is a high frequency in trains. Next to that, Douglas Economics quantifies waiting time as ‘costing’ 1.5 times the driving time, where the driving time is monetized to compare it to other costs. Bakker, P. et al. also mention that waiting time during transfers is quantified in the same way, so 1.5 times ‘as bad’ as driving time. They quote several sources that differ in the quantification of waiting time compared to driving time. One research that they indicate is particularly thorough (Douglas Economics, 2006) finds the following appreciation of waiting time:

[1]ProRail is a Dutch railway company, Ecorys a Dutch research consultancy company and CPB the Dutch national research institute for social economic policy analyses

Frequency (trains/hour)	*Service-interval-time (min)*	*Average waiting time (min)*	*Weight waiting time (peak hour)*	*Weight waiting time (off-peak hours)*
12	5	2,5	2,0	2,0
6	10	5	1,8	1,8
4	15	7,5	1,6	1,5
3	20	10	1,5	1,3
2	30	15	1,3	1,1
0,5	120	60	0,9	0,7

Table A.1: Translation from frequency to travel time based on 'stated preference' research. *Source: Bakker, P. et al. [3]*

In his *Fundamentals of Transportation and Traffic Operations* [5], Daganzo writes that if passengers arrive randomly at a rate of λ passengers per minute, the total waiting time corresponding to the first K headways[2] can be written as

$$W(T) = \sum_{k=1}^{K} \frac{1}{2} \lambda h_k^2$$

where h_k is the length of the k-th headway. This formula follows from Figure A.1, and it comes from the observation that if λ passengers arrive each minute, the first λ passengers wait for one minute on their own, before they are joined by another λ passengers. All 2λ passengers then wait the next minute together, until another group of λ passengers arrives. In the last minute before the train arrives, λ passengers arrive. By the end of the interval all λh_k passengers have collectively waited for $\frac{1}{2}\lambda h_k^2$ minutes and they are picked up by the train. A new group of fresh passengers arrives in the next interval.

$$\begin{aligned}
&\lambda \cdot 1 + 2\lambda \cdot 1 + 3\lambda \cdot 1 + \ldots + (h_k - 1)\lambda \cdot 1 \\
&= \lambda(1 + \ldots + h_k - 1) \\
&= \lambda \left(\frac{h_k}{2} h_k \right) \\
&= \frac{1}{2} \lambda h_k^2
\end{aligned}$$

The total number of passengers serviced in the time T is

$$N(T) = \sum_{k=1}^{K} \lambda h_k.$$

Now the average waiting time for a passenger becomes

$$\frac{W(T)}{N(T)} = \frac{1}{2} \frac{\sum_{k=1}^{K} h_k^2}{\sum_{k=1}^{K} h_k},$$

a number that is independent of λ.

[2]A *headway* is the time or distance between vehicles in a transit system. In this case a headway refers to the time between consecutive buses.

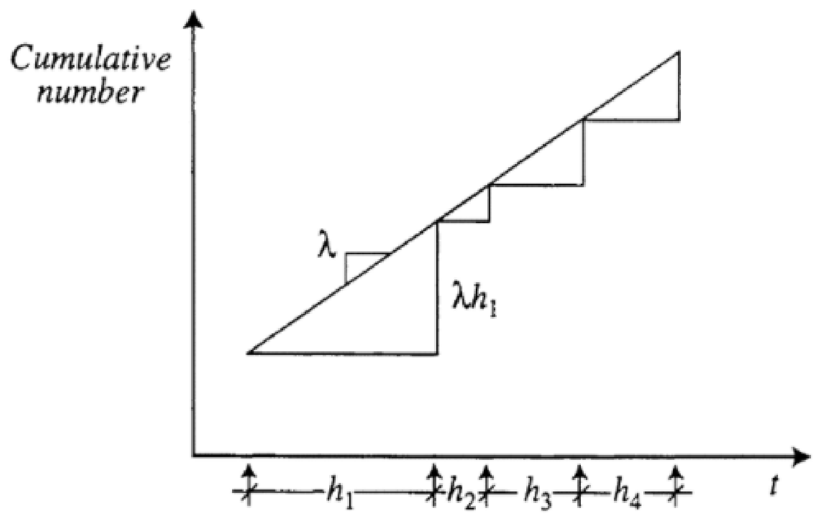


Figure A.1: Cumulative curve of passenger count at a bus stop with a constant arrival rate. Source: Daganzo (2008) [5]

B

Maximizing adherence to PESP frequency constraints vs. maximizing adherence to intuitive frequency constraints

Remember that in finding a cyclic train schedule, the Periodic Event Scheduling Problem imposes frequency constraints on a set of n trains $\mathcal{F}$ in the following way: Between any pair $t, t' \in \mathcal{F}$, the difference in departure time is constricted to be close to either $\frac{60}{n}$, $2 \times \frac{60}{n}$, …, or $(n-1) \times \frac{60}{n}$. 'Close' is defined as: may deviate a margin $m \in \mathbb{N}$ from the perfect interval (both ways). For 2 train series of which one has hourly frequency 2 and the other hourly frequency 1, enforcing both particular and shared frequencies works in the following way:

$$\begin{aligned}
&d(t_1) - d(t_2) \in [20-m, 20+m]_{60} \cup [40-m, 40+m]_{60};\\
&d(t_2) - d(t_3) \in [20-m, 20+m]_{60} \cup [40-m, 40+m]_{60};\\
&d(t_1) - d(t_3) \in [20-m, 20+m]_{60} \cup [40-m, 40+m]_{60};\\
&d(t_1) - d(t_2) \in [30-m, 30+m]_{60}
\end{aligned} \tag{B.1}$$

Note that if we set $m = 0$, (B.1) is equivalent to:

$$\begin{aligned}
|40 - (d(t_1) - d(t_2) + 60)| = 0 \vee |20 - (d(t_1) - d(t_2) + 60)| = 0;\\
|40 - (d(t_2) - d(t_1))| = 0 \vee |20 - (d(t_2) - d(t_1))| = 0;\\
\\
|40 - (d(t_1) - d(t_3) + 60)| = 0 \vee |20 - (d(t_1) - d(t_3) + 60)| = 0;\\
|40 - (d(t_3) - d(t_1))| = 0 \vee |20 - (d(t_3) - d(t_1))| = 0;\\
\\
|40 - (d(t_2) - d(t_3) + 60)| = 0 \vee |20 - (d(t_2) - d(t_3) + 60)| = 0;\\
|40 - (d(t_3) - d(t_2))| = 0 \vee |20 - (d(t_3) - d(t_2))| = 0;\\
\\
|30 - (d(t_1) - d(t_2) + 60)| = 0;\\
|30 - (d(t_2) - d(t_1))| = 0.
\end{aligned} \tag{B.2}$$

However, for any two trains t and t',

$$|40-(d(t)-d(t')+60)| \text{ reduces to } |(d(t')-d(t))-20| \text{ and}$$
$$|20-(d(t)-d(t')+60)| \text{ reduces to } |(d(t')-d(t))-40|.$$

Since norms are commutative, the set of equations (B.2) in turn reduce to

$$\begin{aligned} |40-(d(t_2)-d(t_1))| = 0 \vee |20-(d(t_2)-d(t_1))| = 0 \\ |40-(d(t_3)-d(t_1)+60)| = 0 \vee |20-(d(t_3)-d(t_1)+60)| = 0 \\ |40-(d(t_3)-d(t_2))| = 0 \vee |20-(d(t_3)-d(t_2))| = 0 \\ |30-(d(t_2)-d(t_1))| = 0. \end{aligned} \tag{B.3}$$

For clarity, we adopt the notation $q := d(t_2)-d(t_1)$,$r := d(t_3)-d(t_1)+60$,$s := d(t_3)-d(t_2)$, such as in Figure B.1. We then obtain from (B.3) the following set of equations:

$$\begin{aligned} |40-q| = 0 \vee |20-q| = 0 \\ |40-s| = 0 \vee |20-s| = 0 \\ |40-r| = 0 \vee |20-r| = 0 \\ |30-q| = 0. \end{aligned} \tag{B.4}$$

This way of enforcing regular frequencies seems a bit counterintuitive, as from the perspective of travelers the objective of frequency constraints is to make the intervals as short as possible, so that the passenger needs to wait as little as possible. However, the reason that the constraints are enforced as above is because the order is not fixed. If these constraints are all met, the consequence is that between any consecutive train departures there is a right interval of time.

We saw in Section 1.4 that it is not possible to find a timetable that meets all these constraints at the same time, so not all terms can be zero. Nonetheless, we can calculate how far a timetable is from zero by summing over the terms. Since the intervals are allowed to be close to one of the two intervals, we take the minimum of the deviation of those two as follows, resulting in the following objective:

$$\text{min}\{|40-q|,|20-q|\}+\text{min}\{|40-r|,|20-r|\}+min\{|40-s|,|20-r|\}+|30-q|. \tag{B.5}$$

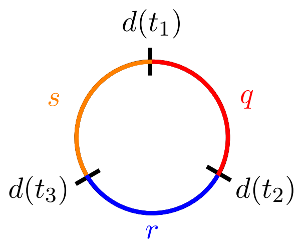


Figure B.1: Three train departures in an hour with intervals q, r and s.

Now assume w.l.o.g. that the order *is* fixed. From the traveler's perspective it makes sense to want all intervals to be as small as possible, as when a traveler arrives at any point in the hour this means that a train is departing soon. The traveler then does not mind if an interval is not exactly perfect, as long as it is not longer. Therefore we consider a new objective that is the additional waiting time:

$$\begin{aligned}\text{max}\{0,q-20\}+\text{max}\{0,r-20\}+\text{max}\{0,s-20\}+\text{max}\{0,q-30\}+\text{max}\{0,(60-q)-30\}\\ =\text{max}\{0,q-20\}+\text{max}\{0,r-20\}+\text{max}\{0,s-20\}+|30-q|.\end{aligned} \tag{B.6}$$

We can now compare the two objectives, which we both want to be minimal. We want to know if they are optimal for the same schedules.

When we minimize the sum of the deviations in (B.5), we obtain the following program:

$$\begin{aligned}\text{minimize}\ \ f(q,r,s) :=\ &\text{min}\{|40-q|,|20-q|\}+\text{min}\{|40-s|,|20-s|\}+\\ &\text{min}\{|40-r|,|20-r|\}+|30-q|\\ \text{s.t.}\ \ &q+r+s=60;\\ &q,r,s\geq 0;\\ &q,r,s\in\mathbb{Z}.\end{aligned} \tag{B.7}$$

Minimizing the 'additional waiting time' in (B.6), results in

$$\begin{aligned}\text{minimize}\ \ g(q,r,s) :=\ &\text{max}\{0,q-20\}+\text{max}\{0,r-20\}+\text{max}\{0,s-20\}+|q-30|\\ \text{s.t.}\ \ &q+r+s=60.\\ &q,r,s\geq 0;\\ &q,r,s\in\mathbb{Z}.\end{aligned} \tag{B.8}$$

The two above versions of the frequency constraints (B.1) and (B.6) are intuitively not the same and we will show that they indeed are not.

C-plex shows that the optimum of (B.7) is 10, and that this is only achieved for **z**$'' = (20, 20, 20)$. C-plex furthermore proves that the optimum of (B.8) is 20. Therefore all optimal solutions to (B.7) are also optimal solutions for (B.8), as (B.7) only has the one optimal solution **z**$'' = (20, 20, 20)$ and filling this in in (B.8) results in

$$g(20, 20, 20) = 20.$$

However, that is the only optimal solution of (B.8) that is optimal for (B.7). Another optimal solution to (B.8), such as **z**$^* = (30, 10, 20)$ is not optimal for (B.7), as that results in

$$f(30, 10, 20) = 20 > 10.$$

We can, however, prove that optimizing the following objective (B.9) is equivalent to optimizing the total additional waiting time in (B.8):

$$\text{minimize}\ \ h(q,r,s) := \text{min}\{|40-q|,|20-q|\}+\text{min}\{|40-s|,|20-s|\}+\text{min}\{|40-r|,|20-r|\}+ \tag{B.9}$$

$$2|30-q| \tag{B.10}$$

$$\text{s.t.}\ \ q+r+s=60.$$

The doubling of $|30-q|$ seems intuitively sound, as the deviation of each of the three intervals within the first set is penalized. It then seems logical to penalize any deviations from the two intervals in the second set. However, PESP does not operate in this way, as it only needs to enforce the difference between one pair of departures instead of the difference between three pairs of departures. We continue below with the formal theorem and the proof of the above assertion.

Theorem B.0.0.1. Let $\mathcal{T} := \{t_1, t_2, t_3\}$ be a set of three trains, of which t_1 and t_2 belong to one train series and t_3 belongs to another. Hence the first train series leaves twice every hour and the second train series leaves once every hour. We enforce particular frequency constraints on $\mathcal{F}_1 := \{t_1, t_2\}$ and shared frequency constraints on $\mathcal{F}_2 := \{t_1, t_2, t_3\}$. Let (B.9) be the program that minimizes the sum of the deviations from the perfect PESP intervals between any pair of trains, and let (B.8) be the program that minimizes the sum of additional waiting times over the different intervals between any two consecutive trains. Then (B.9) is optimal if and only if (B.8) is optimal.

Proof. Assume $\mathbf{z}' = (q', r', s')$ is an optimal solution to (B.9) and $\mathbf{z}^* = (q^*, r^*, s^*)$ is an optimal solution to (B.8). We claim that $20 \leq q' \leq 30$ and $r', s' \leq 20$. Then $h(\mathbf{z}')$ simplifies to

$$\begin{aligned} h(\mathbf{z}') &= \min\{|40-q'|, |20-q'|\} + \min\{|40-s'|, |20-s'|\} + \min\{|40-r'|, |20-r'|\} + 2|30-q'| \\ &= (q'-20) + (20-s') + (20-r') + 2(30-q') \\ &= 80 - s' - r' - q' \\ &= 20 \end{aligned}$$

and $g(\mathbf{z}')$ simplifies to

$$\begin{aligned} g(\mathbf{z}') &= \max\{0, q'-20\} + \max\{0, r'-20\} + \max\{0, s'-20\} + |q'-30| \\ &= q' - 20 + 0 + 0 + 30 - q' \\ &= 10. \end{aligned}$$

Furthermore, we claim that $20 \leq q^* \leq 30$ and $r^*, s^* \leq 20$. Then $h(\mathbf{z}^*)$ and $g(\mathbf{z}^*)$ simplify to

$$\begin{cases} h(\mathbf{z}^*) = 20 \text{ and} \\ g(\mathbf{z}^*) = 10. \end{cases}$$

Since

$$g(\mathbf{z}^*) = g(\mathbf{z}'),$$

any solution $\mathbf{z}^*$ that is optimal for objective (B.9) is also optimal for objective (B.8).

Also, since

$$h(\mathbf{z}^*) = h(\mathbf{z}'),$$

any solution $\mathbf{z}'$ that is optimal for objective (B.8) is also optimal for objective (B.9).

Therefore objective (B.9) is optimal if and only if objective (B.8) is optimal.

We will now prove the claims, all by contradiction.

1. Claim 1:

$$q^* \leq 30.$$

Proof of the claim. Suppose this claim is false, so $q^* > 30$. In this case, deduct 1 from q^*. Then $\max\{0, q-20\}$ becomes 1 smaller, and so does $|q-30|$. The other two terms may change by 1, but this does not matter as this means that the entire objective has become at least 1 better.

2. Claim 2:
$$q^* \geq 20.$$
Proof of the claim. Let $q^* < 20$. We can add 1 to q^*, then the term $|30 - q|$ becomes 1 smaller, while the term max$\{0, q - 20\}$ remains 0. Then r^* or s^* will also become 1 smaller, hence making the entire equation 1 smaller.
3. Claim 3:
$$r^*, s^* \leq 20$$
Proof of the claim. We prove this only for r, as r and s are symmetric. Assume $r^* > 20$. Then $q^* + s^* < 40$. Then deduct 1 from r^*, this makes max$\{0, r - 20\}$ smaller by 1, then add this to the smallest of the two intervals q^* and s^*.
4. Claim 4:
$$q' \leq 30.$$
Proof of the claim. Suppose $q' > 30$. Then $r + s < 30$. If $q' > 40$, it is obvious that deducting at least one point from q' and giving it to r' or s', who will at that point have $r' + s' < 20$, so $r', s' < 20$ will result in a better objective. If $30 < q' \leq 40$, deducting one point from q' will make min$\{|40 - q|, |20 - q|\}$ worse, but the term $|30 - q|$ better, so their improvement and worsening cancel each other out. With the points that you deducted, you can make r' or s' better though, because they had $r' + s' \leq 30$, hence one of them was below 20. Therefore $q' \leq 30$.
5. Claim 5:
$$q' \geq 20.$$
Proof of the claim. Suppose $q' < 20$. Then $r' > 20$ or $s' > 20$. Add 1 to q', then min$\{|40 - q|, |20 - q|\} + |30 - q|$ will become 2 smaller, while r' or s' will become only 1 bigger, so the objective becomes better.
6. Claim 6:
$$r', s' \leq 20.$$
Proof of the claim. Suppose $r' > 20$. If $r' > 20$, $q' + s' < 40$. It is beneficial to subtract 1 from r' and give it to q' or s', whichever is smallest. This also holds for s', therefore $r', s' \leq 20$.

□

We can learn two things from this theorem. First of all, while minimizing the deviation from PESP frequency constraints does lead to a solution that is also optimal for a more intuitive way of measuring how much the intervals deviate from ‘good’ intervals, not all optimal solutions of the intuitive way are found, hence a lot of good solutions get passed by when optimizing PESP frequency constraints.

Secondly, In PESP frequency constraints a set of many trains will have many more intervals that can get penalized than a set of few trains, and this does not scale proportionately to the number of intervals. For example, 8 trains will have $\binom{8}{2} = 28$ intervals to meet, while 3 trains will only have $\binom{3}{2} = 3$ intervals to meet. Furthermore, as we have seen, these are mostly intervals that come from non-consecutive trains, while the goal of frequency constraints is to make the intervals of consecutive trains regular.

C
Stage point abbreviations

Abbreviation	Stage point	Abbreviation	Stage point
Aeg	Amsterdam Erasmusgracht aansluiting	Hnk	Hoorn-Kersenboogerd
Amr	Alkmaar	Hw	Halfweg
Amrn	Alkmaar Noord	Hwd	Heerhugowaard
Ana	Anna Paulowna	Hwd1	Heerhugowaard 1
Ass	Amsterdam Sloterdijk	Kma	Krommenie-Assendelft
Bkf	Bovenkarspel Flora	Kz	Koog aan de Zaan
Bkg	Bovenkarspel-Grootebroek	Nhk	Noordhollandse kanaalbrug
Bll	Bloemendaal	Obd	Obdam
Bv	Beverwijk	Obpa	Overbrakerpolder aansluiting
Cas	Castricum	Pmo	Purmerend Overwhere
Drh	Driehuis	Pmr	Purmerend
Ekz	Enkhuizen	Pmw	Purmerend Weidevenne
Hdr	Den Helder	Rdwa	Amsterdam Radarweg aansluiting
Hdrz	Den Helder Zuid	Sgn	Schagen
Hk	Heemskerk	Sgn1	Schagen 1
Hks	Hoogkarspel	Sptn	Santpoort Noord
Hlm	Haarlem	Sptz	Santpoort Zuid
Hlms	Haarlem Spaarnwoude	Utg	Uitgeest
Hlo	Heiloo	Utg1	Uitgeest 1
Hmt1	Hemtunnel 1	Utg2	Uitgeest 2
Hmt2	Hemtunnel 2	Wm	Wormerveer
Hn	Hoorn	Zd	Zaandam
Hn1	Hoorn 1	Zdk	Zaandam Kogerveld
Hna	Hoorn aansluiting	Zzs	Zaandijk Zaanse Schans

Table C.1: Explanation of stage point abbreviations. Source: Vollebergh [20]