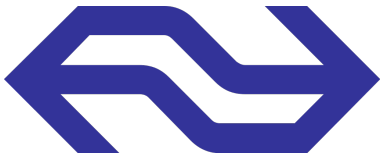


Erasmus School of Economics

In collaboration with Netherlands Railways

Master Thesis in
Quantitative Logistics and Operations Research

SAT modelling for adjusting cyclic railway timetables

Author

Marja van der Wind

(498306mw)

Date

August 10th, 2022

Supervisors

Maaike Vollebergh (NS)

Prof. Dr. D. Huisman (ESE)

Second Assessor

Dr. R. Spliet (ESE)

Abstract

Railway companies often have to deal with construction works on the tracks, for reasons such as maintenance or expansion of the network. These works have a significant impact on the availability of the network so that trains need to be cancelled and retimed. This thesis deals with the problem of adjusting a train timetable in case of construction works. To minimize the hassle experienced by passengers, planners make effort to minimize the number of cancelled trains and retimings. In the literature, researchers tried solving this problem with a Mixed Integer Programming model, which sometimes takes too much computation time. As the distinctive approach of Satisfiability (SAT) modellings works well for creating timetables from scratch, this thesis applies this type of modelling to the Train Timetable Adjustment Problem (TTAP) too. We found that a SAT solver quickly determines whether the problem is feasible, but in case of infeasibility, it takes a long time to decide why this is the case. Therefore, it has so far not defeated a MIP model.

Keywords: SAT, TTAP, TTP, PESP, timetabling, construction works, alternative hour pattern

The content of this thesis is the sole responsibility of the author and does not reflect the views of the supervisor, second assessor, Erasmus School of Economics or Erasmus University Rotterdam.

Contents

Acknowledgment

The writing of this thesis is the final requirement for obtaining my Master's degree in Econometrics and Management Science, with the specialization in Logistics and Operations Research. It is also the result of doing an internship at NS, which I have totally enjoyed. Let me thank some of the people that have helped me to write this thesis.

First, I would like to thank Maaike Vollebergh, who has been my daily supervisor at NS for about five months. Thank you for being patient, kind, and honest with me. Also, I would like to thank Dennis Huisman, my supervisor at Erasmus University. For me, it was very helpful seeing you on a regular basis in the NS-building LVP, and also having informal talks about non-thesis related topics. I enjoyed it that both of you took the time to meet every Monday with the three of us. Thank you for providing me with extensive feedback and helping me improve the quality of my thinking and writing. I would also like to thank Remy Spliet for taking part in my thesis committee.

Furthermore, I want to thank NS for offering me an internship in a nice and safe learning environment, for being flexible and making my start as an intern effortless. Especially, I want to thank team BOS for making me feel welcome and part of the team from the first day I started. I have enjoyed the lunches and drinks, and all jokes during a working day. Especially, I want to thank Adriaan for taking so much care. Also, I thank my fellow interns Susan and Danny for joining the office almost every day and having nice chats.

Finally, I am grateful for my friends and family who supported me throughout my four years of time at university, and also during the writing of my thesis. I especially want to thank my parents, who let me live at home again and were always interested in my progress. And last but not least, I want to thank my driving instructor Angelo for being so patient with me and changing my view on learning and failing, which was of great use for writing my thesis. Learning to drive a motorcycle was a welcome break from programming issues and a great stimulus to stay relaxed during the writing process.

Management Summary

This thesis elaborates on adapting a cyclic railway timetable in case of construction works, also known as the Train Timetabling Adjustment Problem. It is mainly focused at NS, the largest railway operator in the Netherlands. The goal is to find a feasible schedule for one hour, whilst minimizing the hassle for travelers, and taking the reduced infrastructure capacity into account. In such a schedule, it is determined at what exact time each train runs. A timetabling problem includes many requirements, which for years made it too hard to be solved by a solver within reasonable time. Therefore, planners still adapt the timetable manually in case of construction works. To support this process and make it more efficient, we build a decision support system that is based on a satisfiability (SAT) solver. A SAT solver is a solver that is very effective for checking whether a solution exists that adheres to all conditions and presenting one. Recent research shows that using a state-of-the-art satisfiability solver is currently the fastest way to solve the problem of constructing a timetable.

In this thesis, we explain the foundation for SAT modelling and provide a mathematical formulation for the problem. Then, we implement the SAT model and present the results. We also compare these results with other solutions to this problem. Finally, we provide the reader with a summary of our conclusions and recommendations for further research. We find that the use of a SAT solver is promising. Unfortunately, it does not yet beat the results of other solvers such as CPLEX. We attribute the observed deficiency to the obsolete solver that we had at our disposal and to the algorithm that we have used. We are still convinced that the SAT solver could bring a great advantage. First, we need a new, advanced version of the SAT solver, and it should be integrated into the more common solver CPLEX. In this thesis, we propose ideas to accomplish this and share the results of a first implementation.

1 Introduction

The railway network in the Netherlands is extensively being used by several railway companies, running passenger trains, freight trains, and international trains. Netherlands Railways (Dutch: Nederlandse Spoorwegen, NS) is the main passenger railway operator in the Netherlands, providing public transport for hundreds of thousands of travelers each day. A different company owns the infrastructure, namely ProRail. ProRail is the organization that allocates track use among train companies, and is responsible for the infrastructure quality. Therefore, ProRail often plans construction works on the railway tracks to maintain or expand the infrastructure. During these projects, which are mostly scheduled during the weekends, there is a reduced availability of the infrastructure. This makes the original timetable infeasible and requires an alternative timetable for the time the constructions last.

While writing this thesis, the author has experienced extensively what it means if there are construction works on the route you are traveling on. During the summer months, a few kilometers of tracks were replaced between Culemborg and Geldermalsen. It was not possible to run sprinters between Houten and Geldermalsen, and intercities could only run between 's Hertogenbosch and Geldermalsen. In order to work on her thesis at the office, the author was forced to travel further after Geldermalsen by bus to arrive at the NS head office in Utrecht. Although the traveling experience was quite frustrating sometimes, it was also very interesting to experience the problem that this thesis is about. Therefore, this case is also included in this thesis and is presented in Chapter 7.

We refer to the problem of adapting the timetable to a feasible alternative timetable as the Train Timetable Adjustment Problem (TTAP). TTAP is hard to solve, which makes it desirable to design a decision support system that makes use of an efficient solver. Since several years, NS employees have been working on such a solver, and called the project and its model RAAD, which is the Dutch abbreviation of *computations on alternative timetables* (Dutch: rekenen aan alternatieve dienstregelingen). Unfortunately, this solver does not always find a solution in a reasonable amount of time. Hence, the goal of this thesis is to obtain a solver or method that reduces the computation time to a reasonable level.

To solve this problem, it is common to use a Mixed Integer Programming (MIP) formulation of the problem, based on Van Aken et al. (2017a). His model extends the commonly used Periodic Event Scheduling Problem (PESP) formulation. In the literature, several algorithms have been proposed to speed up solving such a MIP, which is further elaborated on in Chapter 3. However, it does not yet solve all instances within reasonable time, suggesting this problem demands an entirely different approach. In this thesis, the problem is therefore modelled as a satisfiability (SAT) problem. This idea is based on results obtained in Vollebergh (2020), where the author uses a SAT solver to solve the problem of constructing a timetable from a line plan, also known by PESP. A line plan only defines on which lines we run services and with what frequency. To the best of our knowledge, modelling TTAP as a SAT problem has not been considered before in the literature. As

we know that solving PESP by a SAT solver works well, and that TTAP is very similar to PESP, modelling it as a SAT problem is very promising, though. This thesis is thus devoted to modelling and implementing a SAT formulation for TTAP, and answers the central research question:

Can a satisfiability solver reduce the computation time of
the RAAD model while still producing high quality solutions?

The relevance of this research is two-sided. Firstly, it contributes to scientific progress on timetabling problems. It especially gives insight on the applicability of SAT modelling for TTAP, and shows if it finds good solutions faster than the MIP solver. This broadens the application field of SAT modelling and deepens the knowledge on how to implement such modelling. On the other hand, this research also contributes practically, in the sense that the Dutch society is highly dependent on public transport. Being able to travel is a necessary condition for many critical processes in the country to remain functioning. Especially when climate goals require more and more people to get rid of their cars, the importance of public transport increases significantly.

We tested the SAT model on two instances that the mentioned papers use for implementing methods on the MIP. Therefore, we are able to compare results. We find that the SAT model can decide on satisfiability within seconds. If the problem is satisfiable, we obtain a solution very quickly. If the problem is unsatisfiable, and we need to cancel some services, it takes the solver quite long to determine which services form a conflict. As our algorithm uses this feature every time we find unsatisfiability in order to cancel another service, it unfortunately does not defeat the RAAD model. However, the SAT solver could be of great use as a part of the RAAD model. We propose to include the SAT solver as a Callback function in every node of the branch-and-cut tree that CPLEX uses to solve the MIP formulation of the RAAD model.

We organize the remainder of this thesis proposal as follows. Chapter 2 provides a detailed description of the problem, and Chapter 3 covers a review of related literature. We present a mathematical formulation of TTAP in Chapter 4, and in Chapter 5 we discuss the encoding of TTAP to a SAT formulation. In Chapter 6 we explain the algorithm that provides problems to the SAT solver and deals with its solutions. Then, in Chapter 7 we provide the reader with the obtained results, and conclusions are drawn in Chapter 8.

2 Problem definition

2.1 Constructing a timetable

Planners at NS construct the timetable from scratch in three steps. First, they make a line plan, in which they determine direct connections and frequencies. In the line plan, they also determine which routes are taken. Thus, routes for trains are fixed and are not changed for the sake of timetable scheduling. Second, they construct a timetable for one hour, containing exact arrival and departure times, as well as connections. Thirdly, they construct the timetable for a whole week, by repeatedly applying the hourly timetable and removing services outside rush hours. Of course, all of this requires expert knowledge and bears its own difficulties. A more detailed description of the planning process at NS can be found in Huisman et al. (2005). As the problem of constructing a timetable became more difficult over the years, it is still mainly a human planning process, in which solvers offer support. In this thesis, the main focus lies on the process of altering the obtained timetable, which is called a Basic Hourly Pattern. This hourly pattern specifies which services run at which times and at which places. It is usual to model this part of the problem as a Periodic Event Scheduling Problem (PESP), which was first introduced in Serafini and Ukovich (1989). The objective of PESP is finding an optimal train timetable that allows for cyclicity. PESP is a macroscopic model in the sense that it does not include local track details. In this thesis, we also choose to take such a macroscopic approach.

In this constructed hourly timetable some conditions must be met, such as satisfying minimal travel times, minimal dwelling times, headway constraints, transfers, equal distribution, and cyclicity and symmetry of the timetable. We illustrate these conditions in Section 2.1.2, and include them in our model constraints as presented in Chapter 4. As does PESP, we consider a macroscopic timetabling problem, so we do not include local details, such as the availability of switches and signals. Therefore, we first elaborate on what a macroscopic approach entails.

macroscopic modelling To model PESP in a way that it is also feasible in practice, one should also take into account local features of the infrastructure. For instance, a train cannot magically jump from one track to the other, but can only switch tracks if there is a switch. Another challenging aspect of timetabling is that most tracks are used by trains of different speeds. The capacity of the track then highly depends on the ordering of the trains on that track and on overtaking options on the tracks. To model this properly, it is necessary to have access to such data on a local level. Assigning trains to platforms and tracks, however, is beyond the scope of PESP, but very interesting for the feasibility of a timetable and thus also for TTAP. However, modelling this requires a microscopic approach, for which also microscopic data is necessary. In Vollebergh (2020) these data were gathered and used for a part of the province Noord-Holland, to attempt including this type of local information in the model. However, as we do not have access to these data for the rest of the country, we apply a macroscopic approach. In a macroscopic approach, we make a general timetable without allocating trains to tracks yet. Also, we only check headway times when there

are only one or two tracks available. If there are more tracks, we do not know which tracks are used by which trains, so we assume that the trains fit in some way. Whether the solution that we come up with also fits from a microscopic perspective, is checked in a later stage.

2.1.1 Concepts

Let us now introduce some commonly used concepts and terminology for timetabling problems in general, as well as timetable requirements. We mainly base the below explanation of important concepts on Mooij (2006).

Infrastructure points In Figure 1, a part of the railway network between Houten and Geldermalsen is schematized. The railway network is composed of nodes and the connections between them. We call nodes infrastructure points (Dutch: dienstregelpunten), which are the points in the infrastructure that are important for making a timetable. The most relevant ones are the stations, which include the tracks and switches that belong to it. Other examples are crossings and bridges. We call the connection (tracks) between two infrastructure points an open track. At an open track, a train cannot change track or direction. Each infrastructure point bears a name as well as an abbreviation that is connected to the name.

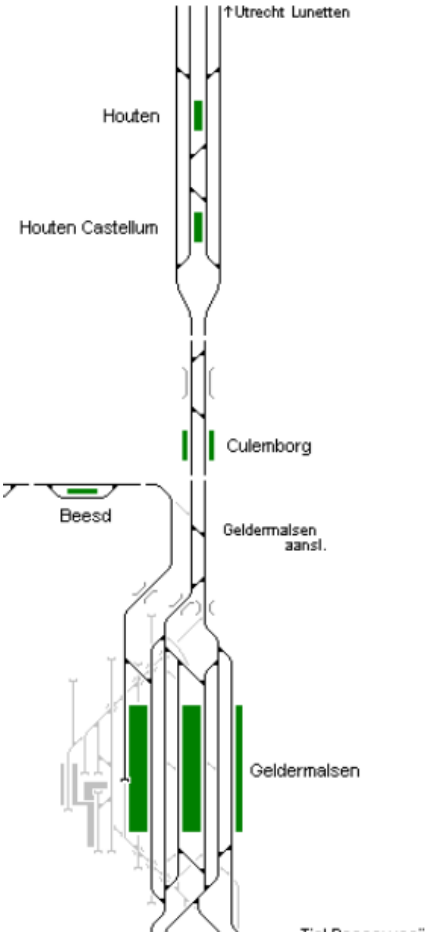


Figure 1: Railway network near Geldermalsen

The Dutch railway network consists of more than 1,000 infrastructure points, of which we see six in the figure, namely Geldermalsen (Gdm), Geldermalsen aansluiting (Gdma), Beesd (Bsd), Culemborg (Cl), Houten Castellum (Htnc) and Houten (Htn). Additionally, we see from the figure that Geldermalsen has five tracks that are connected to platforms, of which two only run in the direction of Beesd. Geldermalsen has two through-going tracks as well, and a collection of switches. Geldermalsen Aansluiting is not a station and has two switches. Culemborg has two switches as well, but also accommodates two platform tracks, which makes it a station. Then, both Houten Castellum and Houten have two platform tracks and two through-going tracks, including some switches to be able to switch between those. Lastly, Beesd has two platform tracks, of which one is a detour track, that only exists at the station, but not at the open tracks before and after Beesd. Although not visible in this picture, every track, switch, and signal has its own code which we do not use or mention further.

Train services We classify all passenger trains by series, and characterize every series by a number. These series consist of a set of services that run at the same fixed route. An example is the series with number 4300, that runs between Almere Oostvaarders (Almo) and Hoofddorp (Hfd) via Duivendrecht (Dvd). In the hourly pattern, these service numbers are multiples of 100. We define odd services as services with a number that is preceded by an odd letter (A, C, E, ...) and go in one direction, and even services in the other direction that have a number preceded by an even letter (B, D, F, ...). However, we do not include deadhead services in the definitions of even and odd series. When planners duplicate the hourly timetable and adapt it to a timetable for the whole day, every train service gets their own unique number (e.g. 4350). They use even numbers for services going in one direction and odd numbers for services going in the other direction. Also, a service that runs one hour later has a number that is four units higher than the first service. The same number applies to services that run the same route at the same time on different days.

The service number also tells which type we deal with. Service numbers between 500 and 40,000 refer to passenger services. Then, service numbers between 70,000 and 90,000 belong to so-called deadhead services that move empty rolling stock to a place where it can be used to accommodate passengers. Freight train companies number their trains according to a different system, and always start with two letters.

Passenger services can be divided in sprinter services and intercity services. The sprinter services dwell at almost every station they pass. Intercity services, on the other hand, only dwell at the larger stations that are used by many travelers. The distinction between these passenger services can not be found from the series number, but they are always part of different series.

2.1.2 Requirements

The adapted timetable should in principle adhere to the same rules and requirements as the original timetable. First, we will briefly explain the conditions below.

Minimal travel times Trains need a minimal time to run between two stations, which is called the technical running time. This time is dependent on distance, the maximum speed, infrastructure restrictions and the train type. However, we do not schedule this technical driving time, as we require an opportunity to drive a little slower for the sake of robustness. Therefore, we define the minimal travel time as the technical driving time plus a supplement (of 8%, mostly). In case any delay occurs, this room can be used to decrease the delay by driving faster. So if a train runs according to the timetable, it runs slower than the maximum technical speed allows.

Minimal dwelling times The minimal dwelling times are given to us as input. In reality, the minimal dwelling time depends on the expected number of passengers getting on or off the train at a station. In turn, this depends on the station, train type and the time of the day. However, due to the cyclic nature of the timetable, the latter is not incorporated in the estimate that is given to us. In our input, the minimal dwelling time is often only one minute. However, specific activities can

add time to the minimal dwelling time, such as combining or splitting carriages. In this thesis, we define a dwelling time of at most one minute as a short dwell, and if the minimum dwelling time is over a minute, we call it a long dwell.

Minimal turning times For instance, in case of turnings at stations, we add extra dwelling time, as not only travelers must get on and off the train, but also the driver needs time to get from one end of the train to the other end. The minimal turning time hence is a specific case of a dwelling time. At some stations, there are exceptions to the regular turning times, which are specified per station.

Minimal headway times Between two trains running in the same direction on the same track, a minimal headway time is compulsory. To arrange this, the tracks are divided in sections of 1,200 to 1,850 meters. If there is already one train in a certain section, no second train can run in that section. At the start of each section, there is a signal that informs the driver about the status of that section. A red signal means that there is already a train in that section, forcing the train to stop. A yellow signal means that the train must slow down because there is a train in the next section, to make sure it can stop in time in case the next signal is red. A green signal means that the upcoming two sections are free of other trains. Thus, basically the headway time depends on the length of the train in the front, the section length and the speed of both trains. However, for planning purposes, we assume a predetermined headway time as in the RAAD model in Maróti and Vollebergh (2021).

Furthermore, we only check for headway times at lines where there are only one or two tracks. In case of one track, it is clear that all trains take that track. For two tracks, we assume that each train takes the one on the right, seen from the direction they headed from. For the reason of our macroscopic approach that we mentioned earlier, we do not make a specific plan where each train runs in case of more than two tracks. As we thus do not know where these trains run, we cannot check for headway time.

Minimal time at a crossing A crossing is what we call a situation in which two trains use the same tracks but do not drive in the same direction. For safety, a minimal time should be planned in between the two trains using the crossing. The minimal margin differs per situation and per combination of trains. For crossover times, we also assume that they are predetermined by the RAAD model.

2.2 Adapting a timetable

In case of construction works, not all routes of series are feasible anymore. For instance, if the infrastructure between certain stations is not available, it is not possible to maintain the frequencies as determined in the original plan. During the creation of the line plan, capacity problems are not taken into account yet. This is first done in the next step of constructing an hourly timetable, so

that the capacity infeasibilities become visible only then. Therefore, adapting the timetable in case of construction works is a task of changing the hourly pattern, but it may also influence the basic plan of which series can run and at what frequency, so that we have to adapt the basic line plan. We call the adapted hourly timetable an Alternative Hourly Pattern (AUP, Dutch abbreviation for Alternatief UurPatroon), which is obtained by solving the Train Timetable Adjustment Problem (TTAP). The objective of TTAP is to minimize the number of cancelled services while deviating as little as possible from the original timetable. At the same time, we still must meet all requirements. The goal to minimize hassle for travelers outweighs the usual desire to prevent possible delays. Also, we do not count rescheduled services as delayed services, as we inform the travelers in advance. To make sure that the adapted timetable sticks to the same requirements as the basic timetable, it is useful to use most of the constraints of PESP for TTAP too, which are further specified in Chapter 4. This insight also induces what the necessary input for solving TTAP is, namely the original timetable, and extra restrictions on infrastructure that are caused by construction works. After we construct the adapted timetable, planners check for microscopic feasibility of the changes, focussing on track use and shunting movements.

2.2.1 Limitations

freight trains To adapt the timetable such that it also meets the added requirements due to restricted availability of the infrastructure, we can perform several actions. In this thesis, we only consider the options to reschedule the services in terms of time, and to cancel services. Rerouting services is also allowed when adjusting a timetable, but it falls outside the scope of this thesis. Mostly, freight trains can be rerouted via routes that other freight trains are also using, as these routes are not used every hour. In this thesis, we consider this option as a free cancellation.

rerouting Moreover, rerouting passenger services seems to be not so beneficial in the Dutch network, as mostly other passenger services already run on those tracks. Therefore, we choose not to incorporate these rerouting options in our model.

bending services Furthermore, usually it is not allowed to bend trains, which means that a service may not run at a lower speed than its maximum speed. NS aims not to bend services as this results in longer travel times. However, in case of construction works we need to increase most travel times anyway to make the schedule fit, so we assume that this requirement is no longer a priority.

3 Literature review

3.1 Periodic Event Scheduling Problem

The model that forms the basis of cyclic railway timetabling was developed by Serafini and Ukovich (1989). In their paper, they describe the well-known Periodic Event Scheduling Problem (PESP) that considers finding a feasible solution for scheduling a set of events under periodic time window restrictions, and they prove that PESP is $\mathcal{NP}$-complete. For the general PESP model, Odijk (1994) proves its $\mathcal{NP}$-completeness, by use of the argument that this problem is a generalization of the Graph Coloring problem. PESP is a macroscopic formulation in the sense that it does not include local details on tracks, switches, and signals. In this thesis, we also consider a macroscopic situation, leaving local planning details to the planners. In Serafini and Ukovich (1989), the authors propose a branch-and-bound procedure for finding feasible solutions. Several other currently available solution methods are Constraint Programming (Odijk, 1994), Genetic Algorithms (Nachtigall, 1996) and of course integer programming techniques. Currently, the most efficient solving approach to solve PESP is to encode it as a satisfiability (SAT) problem and use a state-of-the-art SAT solver (Großmann, 2012).

3.2 Satisfiability modelling

SAT was first introduced by Cook (1971). Although it is usual to write PESP with linear constraints, it is clearly a combinatorial optimization problem, as the number of feasible alternative timetables is finite. For such problems, SAT solvers might be a good solution. Großmann (2012) provides a detailed description of how to translate PESP into a SAT formulation. As TTAP is very similar to PESP, his examples are very useful for our application. The author also shows that it is possible to reduce PESP to SAT in polynomial time. One of the technicalities that he uses, is order encoding of the variables, which was first introduced by Tamura et al. (2009). He shows by empirical results that using this type of encoding makes the solver way faster than with the use of direct encoding. Unfortunately, SAT modelling does not always perform very good. In Abio and Stuckey (2014), the authors state that typically encoding linear constraints to SAT performs poorly in comparison to constraint programming (CP) or mixed integer programming (MIP) solvers. However, they say, some problems contain a mix of combinatorial constraints and linear constraints, where encoding to SAT is highly effective.

Applications of SAT to PESP modelling Gattermann et al. (2016) presents one of the applications of formulating PESP as a SAT problem. In this paper, the authors not only solve PESP by using a SAT approach, but they also include passengers' routes in the SAT model. To consider the most realistic distribution of passengers, they distribute origin-destination pairs over so-called *time slices*, which specifies in which part of the planning period the passenger's journey is supposed to start.

In Vollebergh (2020), the Open-ended Period Event Scheduling Problem (OPESP) is first introduced, as a generalization of PESP. Unlike PESP, that needs fixed track assignments, OPESP is able to incorporate multiple track options per stage. The events are open-ended in the sense that they can take place in several ways, with each event having its own set of constraints. The algorithm consists of an iterative procedure, where each iteration adds an extra track option for each train. They test OPESP using a SAT solver on a slightly simplified version of a part of the Dutch railway network between Amsterdam Sloterdijk (Ass), Enkhuizen (Ekz) and Den Helder (Hdr), in the province Noord-Holland. The SAT solver returns either a feasible schedule or the message that none exists with the given restrictions. According to Van der Knaap (2021), the scalability of this approach is questionable, as every iteration adds many variables, so that the computational size grows rapidly while running the algorithm. Her thesis is hence devoted to finding methods that can use the feedback of the SAT solver efficiently to identify issues, aiming to only add variables when needed.

3.3 Train Timetable Adjustment Problem

In Mooij (2006), the author proposes a model to adjust a timetable for construction works at night. The author considers a very specific situation, in which only part of the tracks are unavailable, such that there is always at least one track available to run trains on. As offering decision support by the use of solvers was fairly new back then, the taken approach is quite simple from the current scientific perspective. The author implements a MIP and uses CPLEX to solve it. He extends his approach by manually stimulating the model to prioritize the branching smartly. One of the main challenges faced in this thesis, is that it is not always possible to meet the requested maintenance time, given that all trains still must run through these track sections at some time. Also, the author proposes an extensive objective function that captures many details, but he does not implement it due to time restrictions.

In the years after, more approaches have been introduced. For instance, in Stut (2009) the author proposes a similar but stochastic model for the case of real-time disruptions, and makes a distinction between primary and secondary disruptions. She defines primary disruptions as having external causes, and secondary disruptions as having internal causes, namely dependencies on other trains in the network. Because the built model was too large to be solved by CPLEX within reasonable time, the author applied Dantzig-Wolfe decomposition in two ways. In the first approach, one part of the problem contains all timetable requirements, and there is a part for each day. In the second approach, every train series number has its own subproblem. Unfortunately, it appears that this approach takes an even longer running time, as the reduction in size does not outweigh the fact that the model has to be re-optimized several times now. Also, the report shows that the Netherlands can not be optimized as a whole, due to too little computer memory, which could of course partially be due to the limited capacity of computers back then.

The adaption of a cyclic timetable that is constructed with PESP modelling, is called the Train Timetable Adjustment Problem (TTAP), and was first introduced by Van Aken et al. (2017a).

The macroscopic model focuses on passenger trains and aims to minimize the deviations from the original timetable, including train cancellations, short-turning and retiming. A few months later, the same authors publish an extension (Van Aken et al., 2017b) in which they focus on solution methods for large instances. First, they apply network aggregation techniques to reduce the problem size. Second, they add extra turnaround activities for short-turned trains to model station capacities. Third, they implement flexibility in short-turning such that they no longer need to fix these choices after the preprocessing step, as opposed to their earlier paper.

Nowadays, it is very common to model our problems as TTAP. First, we review five student's papers on this problem, followed by other papers that discuss a variant of this problem. These are grouped by how they differ from what this thesis is about, all leaving us with good ideas as well as challenges. A more extensive overview of research on this problem can be found in Bešinović et al. (2021).

Students' work For two months, groups of students have worked on the same problem as this thesis as a part of the study Econometrics and Management Science at Erasmus University in Rotterdam. We discuss their ideas briefly here.

In Batelaan et al. (2022), a MIP based on Van Aken et al. (2017a) and Kroon and Peeters (2003) is introduced. For a simple disruption, the model comes up with the optimal solution quite fast. However, for large disruptions (e.g. disruptions at multiple sites, affecting each other), their model runs too slow. Therefore, they implement several heuristics and model adaptions. First, they implement two different methods of assigning trains to tracks. Second, they approach modelling train turnings in several ways. Third, they apply a new way of counting trains on the rails to simplify the track capacity constraints as proposed in Kroon and Peeters (2003), and tighten the bounds on event times so that it is enough to check capacity only at the end points of tracks. Fourth, they reduce the number of binary variables in two ways. First, they implement a type of local search heuristic, making it possible to only consider a part of the Netherlands. Second, they replace the delay constraints by others that are less precise, but do not contain a binary variable to check if the difference between two event times passes the hour boundary. To report the correct delays, these are calculated after solving the model, based on final event times. In their paper, the authors also mention a rolling time window heuristic, which unfortunately was not successful on this problem. Their results are surprisingly good and obtained within very small computation times.

In Veenhof et al. (2022), the authors propose a MIP that is based on Veelenturf et al. (2016). They make the strong assumption that trains always arrive at a station in the same order they departed at the previous station, preventing trains from overtaking. As their model runs very slowly, they decide to also implement a Greedy Randomized Adaptive Search Procedure, also known as GRASP, combined with local search. For the local search, they introduce a constructive component and an improvement component. They define the constructive component as a Binary Programming formulation, that constructs a feasible track allocation for each track in the regular situation, trying

to obtain a feasible disposition timetable from the original hourly pattern. The improvement component is a Variable Neighborhood Search algorithm, for which they use five neighborhoods. Unfortunately, one of the neighborhoods can actually adjust the current solution in only 5% of the situations. Still, their analysis of the results is quite detailed, and gives understandable insights in their findings.

In Doxopoulos et al. (2022), the authors also base their MIP formulation on Van Aken et al. (2017a). As this runs too slow, they also implement a Genetic Algorithm. Their description of creating the initial population, evaluating and selecting the solutions, and performing cross-over and mutations on these, are very detailed. Their algorithm performs well in one case, cancelling no trains and only delaying some. However, for two other cases unfortunately many trains need cancelling, and their Genetic Algorithm is not able to find better solutions within reasonable time. Their sensitivity analysis covers almost every parameter used in the model, not leaving much room for further improving the Genetic Algorithm. It is very interesting to see this method having been implemented for this problem, but due to lack of promising results, we choose not to use this idea here.

In De Best et al. (2022), the authors again base the MIP formulation on Van Aken et al. (2017a). The authors use the Incremental Fix Heuristic to come up with solutions. First, they provide an initialization heuristic. Second, they propose an improvement heuristic. Unfortunately, the descriptions of these heuristics are very concise and therefore hard to reproduce. However, they show that this Incremental Fix Heuristic finds solutions way faster than a CPLEX solver using the MIP formulation does. Although they allow for very little deviation, the obtained results are surprisingly good. In two cases, no trains need to be cancelled, although 20 trains are short-turned in the second case. In another case, the model also performs well, yielding a conclusion that their method may be worth doing more research on. Moreover, if services just alternate between two consecutive stations, they add the option to cancel them. Considering the station capacity, this is a very reasonable choice, that probably affects their solutions positively.

In Van Doorn et al. (2022), the author base the MIP formulation on Van Aken et al. (2017a) again, as well as on Bešinović et al. (2020). To solve the problem, the authors introduce a Greedy Constructive Heuristic that uses a local search algorithm. In this report, the authors describe the methodology very concisely, making it hard to be reproduced. Still, however, they present the results very detailed, providing the reader a very good illustration of how the method performs. One very interesting feature they included in their model, is that they considering detours for passenger trains, to visit stations that are not visited by other trains anymore, due to cancellations. With a maximum allowed delay of 10 minutes, their model runs quite well. However, they can not allow the model to delay trains up to 12.5 minutes, as this yields too high computation times. They also show nicely how the trade-off between redirecting through another station and more delays for other passengers turns out for different choices.

Microscopic models Many researchers model scheduling problems by macroscopic models most often, as those models need a limited amount of data and can be used to solve large-scale and complicated problem instances. In microscopic models there is more consideration of routing, but that requires very much detailed information on infrastructure, such as station layouts, available routes, exact placement of switches and signalling, and rolling stock characteristics. Therefore, these models only apply to smaller problems. For example, in Vansteenwegen et al. (2016), they propose a microscopic approach that is only applicable to small networks, due to the high complexity of network details. As a case study, they use a network of three stations around Brussels, with 80 passing trains per hour in the original timetable. Then, the model runs in a few minutes to simulate 10,000 delay scenarios. In Wüst et al. (2019) the authors introduce a microscopic model on a problem that is called a Track-Choice FPESP (TCFPESP), based on the Flexible PESP as introduced by Liebchen and Möhring (2007). TCFPESP avoids tedious iterations between the process steps of the microscopic capacity planning and mesoscopic capacity planning, in case of infeasibility of the micro-level problem. This is done by implementing flexibility as lower and upper bounds to event times of arrivals and departures.

Extensions In Veelenturf et al. (2016), they propose a macroscopic real-time rescheduling approach, and extend the usual models by including transitions from and to the original timetable. Although they classify their model as macroscopic, they do take into account infrastructure and rolling stock capacities, on a more general view. In Bešinović et al. (2020), the authors extend the model in Van Aken et al. (2017a) to a Freight and Passenger TTAP (FP-TTAP) to introduce routing of freight trains on the network due to possessions. It is capable of finding routes that minimize train journey times, by using the k-shortest path algorithm. For freight trains, it can select alternative freight paths with fewer turnings and non-necessary dwells.

Real-time models Construction works are examples of planned disruptions, but there is also much literature on unplanned disruptions. As these are sudden and not expected by the train operator, a solution should be found in real-time, during operations. In Louwerse and Huisman (2014), the authors make specific suggestions for objective functions for these type of real-time models, and they introduce inventory constraints to determine the disposition timetable. To be more specific, they include a term in the objective function to balance the cancellations in both directions. This way, they distribute the impact of disruptions evenly over the passengers travelling in opposite directions. In Zhu and Goverde (2019), the authors implement a model that is similar to Van Aken et al. (2017a), but newly introduce flexible dwelling and flexible short-turning options. For them, this is an even more important feature, as they consider real-time disruptions. Also, their model is even more complete by including realistic characteristics of the infrastructure, e.g. focussing on networks with both single- and double-track railway lines, whereas most other papers assume single tracks. Another recent idea that looks promising is the idea of implementing a Lagrangian relaxation based decomposition algorithm as was done in Zhan et al. (2021), but it is also meant for real-time disruptions. They update the Lagrangian multipliers by a sub-gradient

method and design a heuristic optimization algorithm to turn infeasible solutions into feasible ones, which then become the upper bounds on the original problem. For a more extensive review of unplanned disruptions, refer to Cacchiani et al. (2014) and Ghaemi and Goverde (2015). The insights of modelling unplanned disruptions can of course help with modelling planned disruptions, but still those are two quite different problems. By contrast, planned disruptions usually take longer, are part of tactical planning, the duration is known, and passengers are mostly aware in advance.

Concluding, much research has been done on timetabling problems already. For PESP, we know that solving by a SAT solver works well. However, we can not yet solve all instances of TTAP within reasonable time. Existing research on TTAP mainly focuses on the MIP formulation of the problem. As TTAP is very similar to PESP, modelling it as a SAT problem is very promising and has not been done before in the literature.

4 Modelling

To model TTAP, it is very common to use PESP constraints. This has been done since the introduction of TTAP in Van Aken et al. (2017a). This chapter covers the Event-Activity Network that is the basis of all this, as well as the PESP framework and the PESP constraints. Then, the application on TTAP is presented, as well as some constraints that are specific to TTAP.

4.1 PESP modelling

PESP was introduced in Serafini and Ukovich (1989) as a model for periodic timetables. First, we define the context and definitions for PESP, to be able to define variables, as well as the necessary constraints that are based on this Event-Activity network graph. Most of the PESP definitions are based on Großmann et al. (2012), and taken from Van der Knaap (2021).

4.1.1 Concepts

First, let us define a cyclic interval. Then, we define activities and events, so that we can finally define the event-activity network, that represents the timetable by a graph.

Definition 4.1. Let $a, b \in \mathbb{Z}$ and $T \in \mathbb{N}$. Let the interval from a to b that only contains integer values be denoted by $[a, b] := \{x \in \mathbb{Z} \mid a \leq x \leq b\}$. Then we call

$$[a, b]_T := \bigcup_{z \in \mathbb{Z}} [a + zT, b + zT] \subseteq \mathbb{Z}$$

the cyclic interval from a to b modulo T, with T as the cycle length of the planning period.

As an hour consists of 60 minutes, it is usual to take $T = 60$. However, to make the schedule more detailed, we consider a time unit of tenths of minutes, so we have $T = 600$.

The Event-Activity Network To create a cyclic timetable, we present each train service by a set of events, namely its departures and arrivals at certain stations, and a set of activities, such as dwelling and running. We plan these events and activities in certain cyclic intervals, as defined above, by applying PESP constraints. We first formally define these concepts here.

Definition 4.2. An event $e \in \mathcal{V}$ is defined as a happening that occurs at a point in time.

Definition 4.3. An activity $a \in \mathcal{A}$ is defined by a mapping that assigns a set of intervals modulo T to a pair of events (e, f) with $e, f \in \mathcal{V}$.

Based on these definitions of events and activities, timetabling in general is based on an *event-activity network*.

Definition 4.4. An *event-activity network* $\mathcal{N}$ is represented by a directed graph $\mathcal{N} = (\mathcal{V}, \mathcal{A})$, where $\mathcal{V}$ is the set of events (vertices) and $\mathcal{A} \subseteq \mathcal{V} \times \mathcal{V}$ the set of activities (arcs).

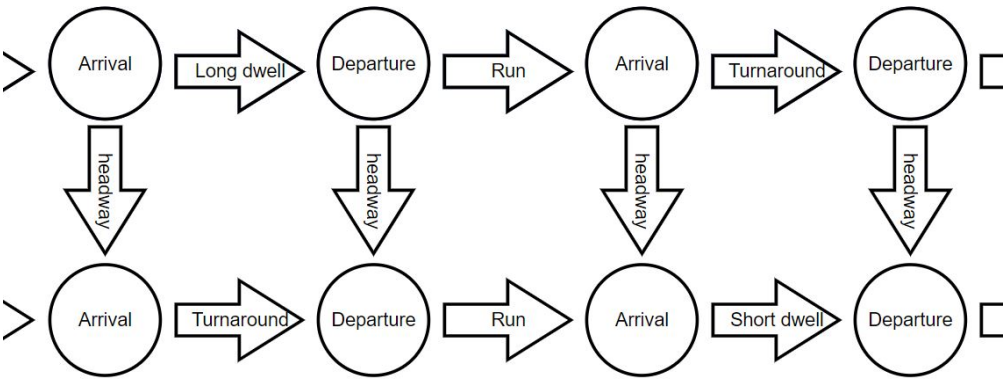


Figure 2: A visualized Event-Activity network

In Figure 2, we schematize an example of an Event-Activity network, by depicting each event as a circle, and activities as arrows. As shown in the figure, each event precedes and succeeds another event, and there is one activity between every two events. In particular, there is always a running arc between a departure and arrival event. Between arrival and departure events, activities such as short dwells, long dwells, and turnarounds can be used.

Definition 4.5. A *schedule* is a function s that assigns to each event a time stamp, $s : \mathcal{V} \to \{0, \dots, T-1\}$. A schedule s is *valid* for an Event-Activity Network $\mathcal{N}$ if and only if all the constraints in $\mathcal{N}$ are satisfied. Two schedules s and d for $\mathcal{N}$ are called *equivalent*, if and only if for all events $e \in \mathcal{V}$ we have that $s(e) \bmod T = d(e) \bmod T$.

Using these building blocks, PESP can be defined as follows:

Definition 4.6. Given an event-activity network $\mathcal{N} = (\mathcal{V}, \mathcal{A})$, the *Periodic Event Scheduling Problem* (PESP) is to find a schedule $d : \mathcal{V} \to \mathbb{N}$ which is valid with respect to all activities $a \in \mathcal{A}$.

4.1.2 Constraints

We now shortly introduce PESP constraints which apply to railway timetabling. PESP constraints ensure that two events are related to each other and happen within a timeframe that is defined by a lower bound and an upper bound. In mathematical terms, we want to achieve the following:

$$\nu_f - \nu_e \in [l_a, u_a]_T \qquad \forall a = (e, f) \in \mathcal{A}$$

which means that the time between event e and event f must be between the lower and upper bound, with the modulo T operator taken into account. We thus define the parameters l_a and u_a as respectively the lower and upper bound of activity a.

First, we describe the properties of PESP constraints as in Van der Knaap (2021). Then, we discuss each constraint and write it in PESP form. We start with three elementary constraints that form the basis of every timetable, namely running, dwelling, and crossing activities, as classified by

Liebchen and Möhring (2007). For these constraints, we define one type of variable, $\nu_e \in [0, T-1]$ as the event time of event $e \in \mathcal{V}$.

Properties of PESP constraints As PESP constraints behave cyclic, it does not matter in which way we order two events. For instance, if we take two arbitrary events (e and f) that take place at different moments, e is always before f, but f is also before e. When we have a constraint that states that the difference between the times of f and e must be in the interval $[l, u]_T$, i.e., $[l, u]_T \in a(e, f)$, we can always rewrite this to a constraint on the difference between the times of e and f. Using that $s(e)$ denotes the time at which event e is scheduled in schedule s, we get:

$$\begin{aligned} & s(f) - s(e) \in [l, u]_T \\ \Leftrightarrow \quad & s(e) - s(f) \in [-u, -l]_T \\ \Leftrightarrow \quad & s(e) - s(f) \in [-u + T, -l + T]_T \end{aligned}$$

Another useful property of PESP constraints is that we can also model a choice between multiple disjoint time intervals. This property is well explained in Van der Knaap (2021), and we provide the well known generalization of this result (Peeters, 2003) in the Lemma 4.1.

Lemma 4.1. Suppose that for some arc $(e, f) \in \mathcal{A}$, we want to impose the constraint

$$d(f) - d(e) \in [l_1, u_1]_T \cup [l_2, u_2]_T \cup \cdots \cup [l_k, u_k]_T,$$

where the k time intervals are ordered and disjoint:

$$0 \leq l_1 \leq u_1 < l_2 \leq u_2 < \cdots < l_k \leq u_k < (l_1 + T).$$

Then the union of these k time intervals is equivalent to the intersection of k periodic time intervals given by the constraints:

$$\begin{aligned} & d(f) - d(e) \in [l_1, u_0]_T \\ & d(f) - d(e) \in [l_2, u_1 + T]_T \\ & \quad \vdots \\ & d(f) - d(e) \in [l_k, u_{k-1} + T]_T \end{aligned}$$

To summarize the problem according to Liebchen and Möhring (2007), a solution of a PESP instance is a node assignment

$$\pi : \mathcal{V} \to [0, T) \text{ that satisfies } (\nu_f - \nu_e - l_a) \bmod T \leq u_a - l_a, \forall a = (e, f) \in \mathcal{A},$$

or in short: $\nu_f - \nu_e \in [l_a, u_a]_T$. From here on, we use this shorter notation for the formal expression above.

Now, we explain how the requirements as described in Section 2.1.2 can be translated to mathematical terms.

Running activities For each running activity that transports a train from one station to another, we set an interval that states how long the activity should last at minimum and at maximum. The lower bound of such an interval is the minimal travel time as defined in Section 2.1.2. We also define the upper bound of a running activity to make sure that the train cannot run too slow, keeping a track occupied for a long time. Also, it is not convenient for travelers if a train runs very slow, as this increases their travel time. We define the corresponding PESP constraint as follows:

$$\nu_f - \nu_e \in [l_a, u_a]_T \qquad \forall a = (e, f) \in \mathcal{R}, \tag{4.1}$$

with $\mathcal{R} \subset \mathcal{A}$ being the set of all running activities.

Dwelling activities For dwelling activities, we distinguish between short and long dwellings. Arcs that belong to short dwelling activities typically have a very small span, as we often fix the dwell time to be zero in case there is neither a junction of tracks, nor a single track. Sometimes, the minimal dwelling time as defined in Section 2.1.2 is somewhat longer, so that we plan a long dwelling. A long dwelling usually takes between one and six minutes, but may take even longer. We define the corresponding PESP constraint as follows:

$$\nu_f - \nu_e \in [l_a, u_a]_T \qquad \forall a = (e, f) \in \mathcal{D}, \tag{4.2}$$

with $\mathcal{D} \subset \mathcal{A}$ being the set of all dwelling activities.

Passing activities If a train passes a station without dwelling, we call the activity a passing and still define an event time ν_e for both the arrival and departure event. The times of these two events are always equal, such that the activity has a lower and upper bound of 0. We then have a PESP constraint:

$$\nu_f - \nu_e \in [0, 0]_T \qquad \forall a = (e, f) \in \mathcal{P}, \tag{4.3}$$

with $\mathcal{P} \subset \mathcal{A}$ being the set of all passing activities.

Headway activities For safety reasons, trains need to maintain a specified minimum time between each other. To quantify this, we define a parameter h_{mn} as the minimal time that a service m must hold from service n with $m \neq n$. We assume that the values of parameter h_{mn} are given. We now use these parameters to model the PESP constraints as follows:

$$\nu_f - \nu_e \in [h_{ef}, T - h_{fe}]_T \qquad \forall a = (e, f) \in \mathcal{H} \tag{4.4}$$

where a is an activity, and the set $\mathcal{H} \subset \mathcal{A}$ is the set of all headway activities.

Potential extra stops Next to these practical requirements that arise in almost all railway timetables, there exist also more specific PESP constraints to specific problems. We cover only a few here, but many of such constraints can be found in Liebchen and Möhring (2007).

First, disjunctive constraints can be modelled by PESP much more elegantly than by most MIPs, as multiple arcs can describe a disjunctive interval without the usually necessary artificial integer variables. We then set the lower bound on the dwell arc to zero, thus modelling for not making an extra dwell. The upper bound on the arc is the sum of (1) the minimal increase b of travel time occurring from braking, dwelling at the station and accelerating, and (2) the maximum dwelling time at the station. Obviously, the effected increase x of travel time must be in the interval $x \in \{0\}_T \cup [b, b+s]_T$. We define the corresponding PESP constraint as follows:

$$\nu_f - \nu_e \in \{0\}_T \vee \nu_f - \nu_e \in [b, b+s]_T \qquad \forall a = (e, f) \in \mathcal{P}_s, \tag{4.5}$$

with $\mathcal{P}_s \subset \mathcal{T} \subset \mathcal{A}$ being the set of all passing activities where we choose to model the possibility to stop.

Unfortunately, there are also timetabling requirements that are not covered by PESP constraints. For instance, we do not always know which trains will run on which tracks. Also, we might not be able to decide a priori which pair of services shall be within the station at the same time, omitting the sequencing constraints between these two services. As a consequence, NS subdivided the problem into two steps, namely the timetabling and the local routing step.

4.2 TTAP modelling

In PESP, we generate a timetable from scratch, evaluating the value of a solution in a self-contained manner. For TTAP, on the other hand, we adapt an available timetable and evaluate it by comparing it to the original one. Adapting a timetable requires different decisions than creating one from scratch. Although we do not want to cancel trains in both problems, in TTAP it may be necessary to obtain a feasible timetable, whereas in PESP it is not allowed at all. Another significant difference between PESP and TTAP, is that for TTAP event times are not equally valued for each event, as in TTAP it is a goal to deviate from the original times as little as possible. In PESP, event times are chosen from scratch and are thus evaluated on their own.

Let us now consider the TTAP model, as solving that problem is our final goal. Fortunately, we can use many constraints of PESP, as adjusting a timetable mainly involve the same safety requirements. Here, we explain the full TTAP model as introduced in Maróti and Vollebergh (2021).

Let us first introduce the used notation, and then present the variables that we need to formally write down the constraints. We define services with indices m, $n \in \mathcal{M}$ with M being the total number of services in the timetable. Then, we define events with indices e, $f \in \mathcal{V}$, with E being the total number of events in the timetable. Each event $e \in \mathcal{V}$ has an original event time π_e. Last, we

define set $\mathcal{A}$ as all regular PESP processes (running, dwelling, passing, headway and turnarounds). For the special cases of optional turnarounds, we define the set $\mathcal{A}_{turn} \in \mathcal{A}$.

Variables

- ν_e is the new event time for event $e \in \mathcal{V}$; and $\nu_e \in [0, T-1]$.
- y_{mk} is 1 if part k of service m is cancelled, where $k \in \{0, 1, \dots \#\text{parts}\}$, and 0 if not; and $y_{mk} \in \mathbb{B}$.
- k_{ef} is 1 if turnaround $ef \in \mathcal{A}_{turn}$ takes place between event e and f, and 0 if not; and $k_{ef} \in \mathbb{B}$.
- p_{ef} is 1 if the later event f has a time ν_f that occurs earlier in the hour than the first event time ν_e, and only exists if $ef \in \mathcal{A}$; and $p_{ef} \in \{0, 1, 2\}$.[1]
- α_e is 1 if the new event time ν_e occurs earlier in the hour than the original event time π_e when the event is delayed, or if new event time ν_e occurs later than the original event time π_e when the event is moved forward in time. This variable exists for every $e \in \mathcal{V}$; and $\alpha_e \in \mathbb{B}$.
- $d_e^+ \geq 0$ is the amount of time event e is delayed relative to the original time π_e, so $d_e^+ = \nu_e - \pi_e$ if $\nu_e - \pi_e > 0$, and 0 otherwise.
- $d_e^- \geq 0$ is the amount of time event e is moved forward in time relative to the original time π_e, so $d_e^- = \pi_e - \nu_e$ if $\nu_e - \pi_e < 0$, and 0 otherwise.
- z_{mk} is 1 if for service m part k is the first part that the service starts running again after cancellation.

Now, we introduce the model as used in Maróti and Vollebergh (2021). First, we provide their objective function. Then we explain all constraints concisely.

$$\sum_m \left(c_m \sum_k y_{mk} + \sum_e p_e \cdot d_e^+ + n_e \cdot d_e^-) \right) \tag{4.6}$$

This objective function consists of two parts. The first part penalizes every part of a service that is being cancelled, and the second part of the objective function penalizes the deviation on event times in the alternative timetable, compared to the original timetable. We define the parameter c_m as the cancelling penalty for service m, and the parameters p_e and n_e as the penalties for positive and negative deviation of the event time of event e. Note that an event is either delayed or put forward in time, so that either d_j^+ or d_j^- is equal to zero, while the other is penalized. If both are zero, no penalty is applied, as the event time has not changed compared to the original one.

[1] A value of 2 is only applicable if the PESP constraint is of the form $0 \leq u < l < T$.

First, we introduce some basic constraints that define lower and upper bounds for event times.

$$\nu_e + l_e \cdot y_{mk} \geq l_e \qquad \forall e \in \mathcal{V} : l_e \leq u_e \qquad (4.7)$$

$$\nu_e + u_e \cdot y_{mk} \leq u_e \qquad \forall e \in \mathcal{V} : l_e \leq u_e \qquad (4.8)$$

$$\nu_e + l_e \cdot y_{mk} + T \cdot \beta_e \geq l_e \qquad \forall e \in \mathcal{V} : l_e > u_e \qquad (4.9)$$

$$\nu_e + (u_e + T) \cdot y_{mk} + T \cdot \beta_e \leq u_e + T \qquad \forall e \in \mathcal{V} : l_e > u_e \qquad (4.10)$$

$$\nu_e + d_e^- + T \cdot \alpha_e + \pi_e \cdot y_{mk} = \pi_e + d_e^+ \qquad \forall e \in \mathcal{V} \qquad (4.11)$$

Constraints (4.7) - (4.8) link the variables y_{mk} to the event times ν_e and its absolute bounds (l_e, u_e). If a service is cancelled, these constraints are satisfied immediately, and for a driving time the event time must be at least its lower bound and at most its upper bound. Constraints (4.9) - (4.10) specifically account for events for which it holds that the lower bound exceeds the upper bound so that we can shift those over the hour boundary. They use the binary variable β_e to apply the modulus function of the cycle length of the planning period. Constraints (4.11) associates the event time ν_e to the original event time π_e, by representing both the delay and forward movement by the d_e^+ and d_e^- variables. Basically, the constraints make sure that $\nu_e = \pi_e + d_e^+ - d_e^- (\text{mod} T)$ in case a service runs, and that $d_e^+ = d_e^- = \alpha_e = 0$ if we cancel a service.

Let us now also introduce some constraints that define lower and upper bounds for activities.

$$\nu_f - \nu_e + T \cdot p_{ef} + l_{ef} \cdot (y_{mk} + y_{nl}) \geq l_{ef} \qquad \forall e, f : (e, f) \in \mathcal{D}, y_{mk} \neq y_{nl} \qquad (4.12)$$

$$\nu_f - \nu_e + T \cdot p_{ef} + (u_{ef} - T + 1) \cdot (y_{mk} + y_{nl}) \leq u_{ef} \qquad \forall e, f : (e, f) \in \mathcal{D}, y_{mk} \neq y_{nl} \qquad (4.13)$$

$$\nu_f - \nu_e + T \cdot p_{ef} + l_{ef} \cdot y_{mk} \geq l_{ef} \qquad \forall e, f : (e, f) \in \mathcal{R} \cup \mathcal{D}, y_{mk} = y_{nl} \qquad (4.14)$$

$$\nu_f - \nu_e + T \cdot p_{ef} + u_{ef} \cdot y_{mk} \leq u_{ef} \qquad \forall e, f : (e, f) \in \mathcal{R} \cup \mathcal{D}, y_{mk} = y_{nl} \qquad (4.15)$$

In Constraints (4.12) - (4.15), event e is in part k of service m, so that variable y_{mk} shows whether event e is cancelled, and event f is in part k' of service m'. These constraints are the PESP constraints that are described in Section 4.1.2. They make sure that the difference between the two activity times does not exceed the bounds of the corresponding activity. If the events (e, f) compose a dwelling activity that is a potential turnaround location, the events e and f belong to different services. Therefore, the possible cancellations of these events are independent, so that the variables y_{mk}, y_{nl} are different. Therefore, Constraints (4.12) - (4.13) are applicable, which make sure that the lower and upper bounds are only checked if none of the two applicable service parts has been cancelled. For other dwelling activities as well as running activities, the two events e and f belong to the same service, such that Constraints (4.14) - (4.15) apply.

To introduce the next constraints, we first define a modulo T function for the upper bound u_{ef} of a variable in case $l > u$ occurs.

$$u_{ef}^* = \begin{cases} u_{ef} & \text{if } l_{ef} \leq u_{ef} \\ u_{ef} + T & \text{if } l_{ef} > u_{ef} \end{cases}$$

We also define an auxiliary variable y_e^* as follows:

$$y_e^* = \begin{cases} y_{mk} & \text{if event e is an arrival and part k of service m ends at the station of event e} \\ y_{mk'} & \text{if event e is a departure and part k' of service m starts at the station of event e} \\ 1 & \text{otherwise} \end{cases}$$

Now, we can define the other PESP constraints. First, we provide constraints that deal with turnarounds.

$$\nu_f - \nu_e + T \cdot p_{ef} \geq l_{ef} \cdot k_{ef} \qquad \forall e, f : (e, f) \in \mathcal{T} \tag{4.16}$$

$$\nu_f - \nu_e + T \cdot p_{ef} \leq u_{ef}^* + (T-1) \cdot (1 - k_{ef}) \qquad \forall e, f : (e, f) \in \mathcal{T} \tag{4.17}$$

$$\sum_f k_{ef} \geq y_e^* - y_{mk} \qquad \forall (e, f) \in \mathcal{T} \tag{4.18}$$

$$\sum_e k_{ef} \geq y_e^* - y_{mk} \qquad \forall (e, f) \in \mathcal{T} \tag{4.19}$$

$$y_{mk} + \sum_f k_{ef} \leq 1 \qquad \forall (e, f) \in \mathcal{T} \tag{4.20}$$

$$y_{nl} + \sum_e k_{ef} \leq 1 \qquad \forall (e, f) \in \mathcal{T} \tag{4.21}$$

$$\sum_f k_{ef} \leq y_{mk'} \qquad \forall (e, f) \in \mathcal{T}, \quad \exists y_{m'k'} \tag{4.22}$$

$$\sum_e k_{ef} \leq y_{nl'} \qquad \forall (e, f) \in \mathcal{T}, \quad \exists y_{m'k'} \tag{4.23}$$

Here, event e belongs to part k of service m and event f belongs to part n of service l, and the successory event of event e belongs to part k' of service m if such an event exists. Similarly, the preceding event of event f belongs to part l' of service n.

Constraints (4.16) - (4.17) are the PESP constraints for turnaround activities, as described in Section 4.1.2. They make sure that there is enough time to turnaround and also account for the upper bound on these activities. As the cancel variables y_{ef} are linked to the turnaround variables k_{ef} in Constraints (4.18) - (4.19), it is not necessary to include them here. The variables k_{ef} form a matching in a bipartite graph of all start and end events. In Constraints (4.18) - (4.23), we make sure that this matching covers every node of the graph, putting a lower bound on the matching variables k_{ef} with Constraints (4.18) - (4.19), and an upper bound with Constraints (4.20) - (4.23).

Specifically, Constraints (4.18) and (4.20) - (4.22) deal with the end events, whereas Constraints (4.19) and (4.21) - (4.23) deal with the start events. This way, each event indeed has exactly one partner, and is not matched if its part is cancelled.

We also outline general constraints that apply to activities other than the discussed ones.

$$\nu_f - \nu_e + T \cdot p_{ef} + l_{ef} \cdot (y_e + y_f) \geq l_{ef} \qquad \forall e, f : (e, f) \in \mathcal{A} \setminus (\mathcal{R} \cup \mathcal{D} \cup \mathcal{T}) \tag{4.24}$$

$$\nu_f - \nu_e + T \cdot p_{ef} + (u_{ef} - T + 1) \cdot (y_e + y_f) \leq u_{ef}^* \qquad \forall e, f : (e, f) \in \mathcal{A} \setminus (\mathcal{R} \cup \mathcal{D} \cup \mathcal{T}) \tag{4.25}$$

Constraints (4.24) - (4.25) deal with the lower and upper bounds of all other activities than the three that we have covered above. These other activities include passing activities and several types of headway activities.

Let us now introduce constraints that link the cancelling variables, and make it possible to cancel parts of a service.

$$y_{mk} \leq z_{m(k+1)} + y_{m(k+1)} \qquad \forall m, k \text{ with } k \in \{1, \ldots, \#\text{ parts in } m\}, m \in \mathcal{M} \tag{4.26}$$

$$\sum_{k=1}^{\#\text{parts}-1} z_{mk} \leq 2 + y_{m0} \qquad \forall m \in \mathcal{M} \tag{4.27}$$

Constraints (4.26) - (4.27) make it possible to cancel parts of a train service by counting how much parts are cancelled in each service and restricting that to two per service, unless the front part is cancelled. In any case, the service can be split in at most three parts. The parts are defined as sequences of events between possible turning stations. Several parts can be cancelled independently of each other. Constraints (4.26) guarantee that part $k+1$ is classified as a starting point for a service if part k is cancelled and part $k+1$ is the first running part, with corresponding start event $z_{m(k+1)}$. Constraints (4.27) ascertain that a service consists of at most three cohesive parts by bounding the number of new start events to two, unless the first part of the service was cancelled, so that three new start events also result in three parts.

Last, we include constraints that address overtaking scenarios.

$$p_{ef} + p_{e'f'} + p_{ee'} + p_{ff'} = 2(w_{ee'ff'} + v_{ee'ff'}) \qquad \forall e, f : (e, f), (e', f') \in \mathcal{R}, (e, e'), (f, f') \in \mathcal{H} \tag{4.28}$$

with $w_{ee'ff'} + v_{ee'ff'} \in \mathbb{B}$. Constraints (4.28) deal with overtaking situations, and are inspired by Zhang and Nie (2016). At sections where there are at most two tracks available, we assume that trains cannot overtake. Then, the sum of the four activities as mentioned in the left-hand side of the equation must be either 0, 2, or 4. In other words, it must be a multiple of two. A proof and examples of these constraints can be found in Zhang and Nie (2016).

5 Satisfiability formulation

Modelling a problem as a Satisfiability (SAT) problem was first suggested in Cook (1971). Still, this approach proves to be useful in many fields, such as logic, graph theory, computer science and operations research. Vollebergh (2020) presents a SAT formulation as an effective tool for a PESP model based on the findings of Großmann (2012). As TTAP is based on PESP, we transform TTAP into SAT too. However, before reducing our problem to SAT, let us first introduce the basic concepts and definitions of SAT modelling.

5.1 SAT encoding

To introduce SAT properly, we first provide some background in propositional logic, inspired by Großmann (2012) and Van der Knaap (2021). To fit the definitions to this thesis, we have changed some notation, but we did not make any substantive changes.

Definition 5.1. The *alphabet of propositional logic*, denoted by Σ_{SAT}, consists of a countably infinite set of propositional variables $\mathcal{P} = \{p_1, p_2, \dots\}$, the brackets "(", and ")", as well as the two binary connectives $\wedge$ and $\vee$, and the unary connective $\neg$.

The connectives in $\{\wedge, \vee, \neg\}$ are called conjunction *and*, disjunction *or*, and negation *not*, respectively.

Definition 5.2. A finite string F that only consists of letters of the alphabet Σ_{SAT} is called a *propositional formula*, if and only if it fulfills one of the following properties

1. $F = p$, for some $p \in \mathcal{P}$, which gives $\mathit{length}(F) = 1$ or
2. $F = \neg G$, for some propositional formula G, which gives $\mathit{length}(F) = 1 + \mathit{length}(G)$ or
3. $F = (G \circ H)$, for some propositional formulas G and H , and some binary connective $\circ$, which gives $\mathit{length}(F) = 3 + \mathit{length}(G) + \mathit{length}(H)$.

We denote the set of all propositional formulas under the alphabet Σ_{SAT}, by $\mathcal{L}(\Sigma_{SAT})$.

Definition 5.3. A propositional formula $L \in \mathcal{L}(\Sigma_{SAT})$ is called a *literal*, if and only if $L = p$ or $L = \neg p$, with $p \in \mathcal{P}$.

Definition 5.4. A propositional formula $C \in \mathcal{L}(\Sigma_{SAT})$ is called a *clause* if it is a disjunction of literals. Hence, with $n \geq 0$

$$C = L_1 \vee L_2 \vee \cdots \vee L_n)$$

where L_i ($i \in \{1, \dots, n\}$) are literals. Note that a clause can be empty.

Definition 5.5. A propositional formula $F \in \mathcal{L}(\Sigma_{SAT})$ is in *conjunctive normal form* (CNF) if it is a conjunction of clauses. Thus, with $m \geq 0$

$$F = C_1 \wedge C_2 \wedge \cdots \wedge C_m)$$

where C_i are clauses.

Most SAT solvers only take input that is in CNF, as this form has modelling advantages that increase the speed of the solver. For instance, all clauses of the formula must be satisfiable independently. Thus, if the solver finds one clause that is unsatisfiable, it stops searching in the direction of that interpretation. To get an interpretation of the variables, we use the following definitions from Ebbinghaus et al., 2013, adapted to our notation.

Definition 5.6. An *assignment* is a map $\beta : \mathcal{P} \to \{true, false\}$ of the set of variables into the domain $\{true, false\}$.

We define the following mapping to evaluate a formula.

Definition 5.7. Let $F \in \mathcal{L}(\Sigma_{SAT})$ be a propositional formula. Then an *interpretation* is a mapping $I : \mathcal{L} \to \{true, false\}$, with

$$F^I = \begin{cases} \beta(p), & \text{if } F = p \text{ for some } p \in \mathcal{P} \\ \neg(G^I), & \text{if } F = \neg G \text{ for some } G \in \mathcal{L}(\Sigma_{SAT}) \\ G^I \circ H^I, & \text{if } F = G \circ H \text{ for some } G, H \in \mathcal{L}(\Sigma_{SAT}), \circ \in \{\wedge, \vee\}. \end{cases}$$

Note that the formula induction used in this definition is well-defined, as from Definition 5.2 it follows that the length of G or the combined length of $G\&H$ is strict smaller than the length of F.

Using the definitions described above, the SAT problem can be defined as follows.

Definition 5.8. Let $F \in \mathcal{L}(\Sigma_{SAT})$ be a propositional formula in CNF. The *SAT problem* is the problem of deciding whether

1. F is satisfiable if there exists an interpretation I such that $F^I = true$.
2. F is unsatisfiable if for all interpretations I we have $F^I = false$.

Now that we have defined the basic definitions of SAT, we can model TTAP as such a formulation. To model TTAP as SAT, we use order encoding to transform most parts of the usual TTAP formulation into a SAT formulation, as this is a very convenient way of modelling cyclic variables and constraints. Order encoding was first introduced in Tamura et al., 2009, where the Constraint Satisfaction Problem was modelled as a SAT problem.

5.2 Variables

Let us define the most important variables in the SAT formulation, namely the times on which events will take place in the adapted timetable. In the PESP model, the variables ν_e present the event times. These variables are in a finite domain, so that $dom(\nu_e) = [l_e, u_e] \subset \mathbb{N}$. In the SAT formulation, we can only make use of boolean variables. Hence, we define a function that transforms these variables with a finite domain into boolean variables, using the definition in Großmann (2012).

Definition 5.9. Let $\nu_e \in \mathcal{V}$ be a variable with a domain $dom(\nu_e) = [l_e, u_e] \subset \mathbb{N}$. Then we define the order encoding function as

$$encode_{ordVar} : \nu_e \mapsto \bigwedge_{m \in [l_e+1, u_e-1]} (\neg x_{m-1}^{\nu_e} \vee x_m^{\nu_e})$$

with $x_m^{\nu_e} \in \mathcal{P}$ for all $m \in [l, u-1]$.

The order encoding function of transforming integer variables into boolean variables uses variables $x_m^{\nu_e}$ that are true if the variable ν_e is smaller than or equal to the value m. Given an interpretation I, variables $x_m^{\nu_e}$ are specified as follows:

$$x_m^{\nu_e} = true \Leftrightarrow \nu_e \leq m$$

Definition 5.10. Let ν_e be a variable with $dom(\nu_e) = [l_e, u_e] \subset \mathbb{N}$, which is encoded by the function $encode_{ordVar}$, and I an interpretation. Then for a given interpretation I there exists a $k \in [l_e, u_e]$ with

$$\nu_e = \begin{cases} l_e, & \text{if } x_{l_e}^{\nu_e} = true,\ k_e = l_e \\ k_e, & \text{if } x_{k-1}^{\nu_e} = false \wedge x_k^{\nu_e} = true,\ k \in [l_e + 1, u_e - 1] \\ u_e, & \text{if } x_{u_e-1}^{\nu_e} = false,\ k = u_e \end{cases}$$

This definition is a formal way of writing that the index of the first variable that takes the value *true* is the value of the original variable. Due to this specific interpretation, we need one less boolean variable $x_m^{\nu_e}$ than the domain interval width. If all variables take value *false*, the upper bound value of the variable $x_{l_e}^{\nu_e}$ applies.

If we find that the problem is unsatisfiable, this is caused by multiple clauses that contradict each other, in the sense that there is no solution available that satisfies these clauses at the same time. We define a Minimal Unsatisfiable Subset (MUS) as the smallest subset of clauses that are combined unsatisfiable.

If we find a satisfiable solution to the problem, however, we need to transform the SAT variables back to event times, for which we use Definition 5.10.

5.3 Constraints

As we have now introduced SAT, we here explain how we model TTAP as a SAT formulation. First, we explain how we transform PESP constraints as introduced in Section 4.1.2 into SAT. Then, we illustrate how we include the other constraints that could not be modelled as PESP.

5.3.1 PESP constraints

The first type of constraints that we transform to SAT, are PESP constraints. This category of constraints can all be transformed using the same function, as they are all of the same form. The

constraint encoding that we use here, describes the infeasible region that is caused by the area outside the interval of the PESP constraints. PESP constraint do not allow certain combinations of values. For instance, two event times of a running activity may lie too close to each other, violating the minimal travel time (lower bound). Thus, to describe the feasible region in SAT, we use clauses to eliminate combinations that violate the requirements as described in Section 2.1.2. More explanation and some visual examples are given by Van der Knaap (2021).

To describe how to encode a PESP constraint, we use the ideas from Vollebergh (2020) by using vertical line segments to describe the feasible region, which was inspired by Großmann (2012) who uses squares to do so. For the next definition, we define $\mathcal{I}_T$ as the set of all intervals modulo T. The order encoding function of a PESP constraint is then given by

Definition 5.11. Let (e, f) be two periodic events that form activity $a = (e, f) \in \mathcal{A}$, and let c be the PESP constraint corresponding to activity a. Also, $s \times [t_1, t_2] \in \zeta(e, f, c)$ is a line segment in PESP Constraint c. Then

$$encode_{ordCon}(e, f, c) \mapsto \bigwedge_{\{s\}\times[t_1,t_2]\in\zeta(e,f,c)} \left(x^{\nu_e}_{s-1} \vee \neg x^{\nu_e}_{s} \vee x^{\nu_f}_{t_1-1} \vee \neg x^{\nu_f}_{y_2}\right)$$

Definitions 5.9 and 5.11 define the functions that we need to encode the complete PESP problem. We use Figure 3 to explain this formula. On the horizontal axis we plot the event time of event e, and on the vertical axis we plot the event time of event f.

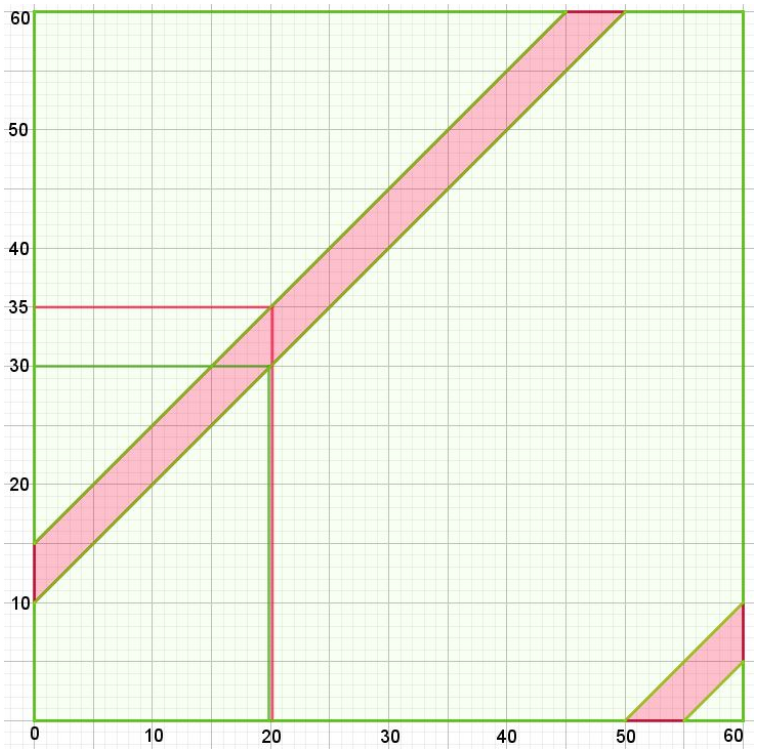


Figure 3: Model a PESP constraint with SAT variables

Example 5.1. To describe the infeasible region in CNF, using Definition 5.11, we need T such formulas in SAT form. We explain one of them with help of the figure, and the other $T-1$ work similar. For simplicity, we take $T = 60$ here.

In words, we want to say the following: if event 1 takes place at minute 20, event 2 may not take place between minutes 30 and 35. In other words, we say: either event 1 cannot take place at minute 20, or event 2 cannot take place between minute 30 and 35. If one of these statements is true, we are sure that we excluded this line in the feasible region. We can write this in SAT, using the four SAT variables that we see in Definition 5.11. For the first part, we use $x_{19}^{\nu_e} \vee \neg x_{20}^{\nu_e}$. So either $\nu_e \leq 19$ or that $\nu_e \not\leq 20$, corresponding to the vertical green and red line, respectively. If one of these is true, then the event time of event e is not at minute 20. For the second part, we apply $x_{29}^{\nu_f} \vee \neg x_{35}^{\nu_f}$. This means that either $\nu_f \leq 29$ or that $\nu_f \not\leq 35$, corresponding to the green and red line, respectively. This means that the event time of event f cannot be between minute 20 and 35. Thus, if event e takes minute 20, we describe the infeasible region as follows: $x_{19}^{\nu_e} \vee \neg x_{20}^{\nu_e} \vee x_{29}^{\nu_f} \vee \neg x_{35}^{\nu_f}$. Of course, to describe the full feasible region, we do this for all possible event times ν_e.

Definition 5.12. Let $\mathcal{N} = (\mathcal{V}, \mathcal{A})$ be an event-activity network. Then the order encoding of network $\mathcal{N}$ to SAT is defined by applying functions $encode_{ordVar}$ to all variables ν_e and $encode_{ordCon}$ to all PESP constraints.

5.3.2 Other constraints

Not all constraints for TTAP can be modelled as PESP constraints. In Section 4.2 we defined Constraints (4.18) - (4.23) that consider turnarounds. Also, we modelled cancelling (parts of) train services inside PESP constraints, which make them no longer purely PESP. These constraints are not written in PESP form, and should either be omitted from the model, be accounted for outside the SAT model, or manually be rewritten to SAT.

Cancelling (parts of) services The most important part of the problem that we do not model in SAT via PESP constraints is the possibility of cancelling services. This possibility which was included in all PESP constraints and the objective function through the variables y_{mk} which represents that PESP constraints no longer need to hold if we cancel the corresponding service. As there is no objective function in a regular SAT model, an incentive for not cancelling services can not be included well. Therefore, the SAT model would always cancel as many services as it is allowed to, as this makes many clauses redundant. That way, cancelling services makes the problem very easy to solve, but mostly does not yield a good solution. To counter this incentive that is opposite of what we want, it is probably best to decide on which (parts of) services to cancel outside the SAT model. The algorithm that feeds problems to SAT thus decides which parts of the service can be cancelled, and is of course prone to cancelling as few as possible.

Turnaround constraints For the sake of conserving the rolling stock flow, we need to tie the end of a service to the start of a successive service. We do this by turnaround constraints, that turn

rolling stock from one service into another. Which services can turn into each other is dependent on the type of rolling stock that we use for each of the services and on other requirements. In this thesis, we assume that it is given which train services can turn into each other.

To model these turnarounds, we need something that is more sophisticated than the regular PESP constraints, as the MIP formulation leaves room to choose between several turnaround options for each service, and match them between services. On the one hand, the fact that we decide which services are cancelled outside the SAT model significantly reduces the number of turnaround options. Namely, in the model in Maróti and Vollebergh (2021), PESP constraints for turnarounds are included for every turning station where a service can be cancelled partly. However, in our case we know whether a service (part) is cancelled, so that the end-points of each service are evident, and we only need to add the constraints that apply to these turnaround options. So although the model still chooses what services can turn into each other, the number of possibilities has decreased. To make a SAT-solver choose between several options, we can use the PESP constraints that check for timing issues as building blocks.

First, define set $\mathcal{O}$ as all outgoing services and set $\mathcal{I}$ as all ingoing services. Then, we also define a propositional formula t_{oi} as the SAT encoding that belongs to the PESP constraint of the turnaround between services $o \in \mathcal{O}$ and $i \in \mathcal{I}$.

$$\bigvee_{i \in \mathcal{I}} \left(t_{oi} \bigwedge_{o' \in \mathcal{O} \setminus o} \left(\bigvee_{i' \in \mathcal{I} \setminus i} t_{o'i'} \right) \right) \qquad \forall o \in \mathcal{O} \tag{5.1}$$

$$\bigvee_{o \in O} \left(t_{oi} \bigwedge_{i' \in \mathcal{I} \setminus i} \left(\bigvee_{o' \in \mathcal{O} \setminus o} t_{o'i'}) \right) \right) \qquad \forall i \in \mathcal{I} \tag{5.2}$$

Constraints (5.1) make sure that if one turning fits for an outgoing service, then all other outgoing services also fit to at least one other ingoing service. Constraints (5.2) do the same for ingoing services. Together, these constraints make sure that not only every service has at least one other service that it can turn into, but also that the remaining services can turn into each other. Also, each service must be turned upon at least once, and then the remaining services must also have other services that fit in a matching together. Please note that this formula is a clause, as it is a conjunction of multiple propositional formulas in CNF. However, as a logical *and* is nested in logical *or* parts, this conjunction of clauses is not in CNF. In Chapter 6 we further elaborate on how we deal with this.

6 Algorithm

In this chapter, we apply a Satisfiability (SAT) solver to TTAP, as it has proven to be very useful for PESP. The SAT solver does not take into account an objective value, but only checks whether there exists a feasible solution. Therefore, the model lacks the urge to not cancel too many services or run services at entirely different times, so we use a different algorithm to take care of optimization that we call the Feedback Loop.

6.1 General approach

First, we create PESP constraints as described in Chapter 5. Based on the events used in these constraints, we make a set of events based on the timetable, and extract the corresponding current event times from the timetable. For all events, we make $T+1$ SAT variables corresponding with T minutes and 1 dummy variable. Then, after we write all constraints in CNF form, the SAT solver solves the problem. If the SAT solver reports that the problem is satisfiable, we describe the solution in terms of the event time as proposed by the SAT solver for each event, which is equivalent to an adapted timetable.

If the SAT solver cannot find a feasible solution, it reports a MUS. Using the MUS, we adapt the problem slightly and again ask SAT if it can find a solution. We apply this procedure iteratively, and we call it the Feedback Loop, which we describe now.

6.2 Feedback loop approach

Now, we present how we solve TTAP by using the SAT formulation. We explain the procedure in words as well as in a graphical representation in Figure 4.

We provide input in CNF to the SAT solver that contains the problem formulation. The SAT solver then solves the problem. If the problem is satisfiable, we extract the event times from the solution and present the new timetable. If the problem is unsatisfiable, the SAT solver cannot find any interpretation under which the propositional formula is true, and the SAT solver provides us a MUS. A MUS is the smallest subset of clauses that form a contradiction, and deleting one of them would solve the conflict. We then translate back the clauses to their corresponding PESP constraints to find out which events and services are causing problems. If at least one of the PESP constraints in the MUS is a turnaround

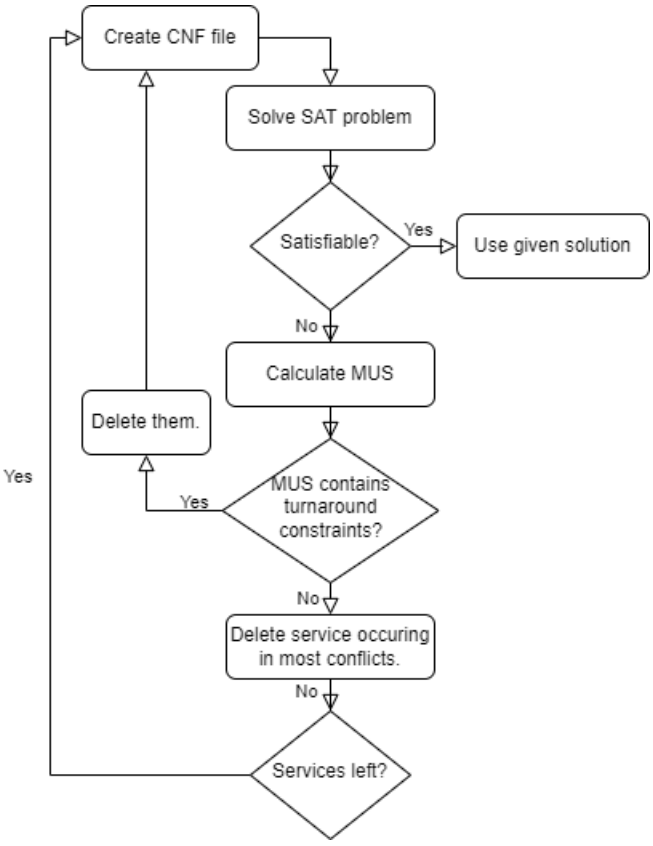


Figure 4: A visualization of the Feedback Loop algorithm

constraint, we check if we can delete it. If deleting it still allows for at least one possible combination of turnaround for all services, we delete the constraint. If there is no turnaround constraint that we can delete to solve the conflict, we calculate for each service how many conflicts they contribute to, and check which service appears in the most conflicts. Finally, we cancel the service that occurs in most conflicts and delete its corresponding constraints. Sometimes, we cancel more services, which we explain in the next paragraph. We rerun the SAT solver with the remainder of the constraints and keep repeating until we find that the problem is satisfiable, or run out of services. Both situations return a feasible solution.

Number of cancelled services per iteration For the feedback loop, we introduce a parameter for the number of cancelled services per iteration. The default setting is to cancel one service per iteration. To try to speed up the process, we can also choose to cancel two services per iterations, or even the whole series. If we cancel two services, we cancel both the service with the most conflicts as well as the service that it turns upon. If multiple turnings are possible, then we choose among these services by finding which service contributes to the most conflicts. If this does not help either, we choose a random turning service. Moreover, we can choose to cancel the whole series. Thus, if the service that occurs in most conflicts is part of a series that consists of four services, all four are cancelled at once.

Turnaround constraints Regarding the PESP constraints, a combination as formalized in (5.1) and (5.2) must hold. As it appears hard to write these turnaround constraints in CNF, we choose a sequential implementation for the turning constraints that represent the same idea. We add all turnaround PESP constraints t_{oi} to the model. If the turning constraints are part of a conflict, we remove the corresponding turning t_{oi}. We repeat this process until either the turning subproblem becomes feasible, or there is a service for which all turnings are deleted. In the first case, we continue checking all other PESP constraints for feasibility and let SAT search for a solution. In the latter case, we return that the problem is infeasible.

7 Results

In this chapter, we first describe the test instances that we use to evaluate the proposed SAT model. Then, we present the results, comment on them, and compare them to work of others.

7.1 Test cases

The two cases are based on real-life situations for the Dutch railway network in 2022. The construction works have taken place in the summer months of 2022.

7.1.1 Case 1

In case 1, construction works take place between Oss (O) and Wijchen (Wc). Between these two stations, only one track is available. Between Oss and Ravenstein (Rvs), there were two tracks available originally, but between Ravenstein and Wijchen there is a part that already had one track available. In this case, only partial blockades are present. As can be seen from Figure 5, originally there run two series between Oss and Wijchen, namely Intercity 3600 and Sprinter 6600. Both of these are series that run twice an hour in each direction. Also, we require some freight trains to be rerouted via a standard rerouting plan. All freight trains that use the route from 's Hertogenbosch (Ht) to Arnhem (Ah) via Nijmegen (Nm) now run via Betuweroute (Brmet).

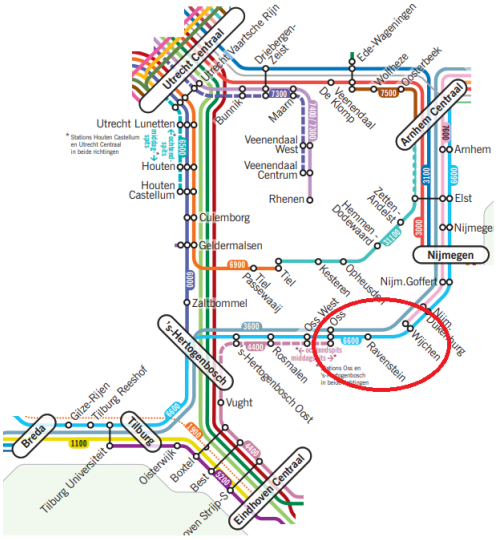


Figure 5: Constructions in case 1

7.1.2 Case 2

In case 2, constructions take place near Culemborg (Cl) as visualized in Figure 6. Only two of the four tracks are available between Houten (Htn) and Houten Castellum, but between Houten Castellum and Geldermalsen (Gdm) there are no tracks available. In this case, we thus deal with partial blockage as well as full blockage. Typically, many services run on these tracks, as these connect the south of the Netherlands to Utrecht, which is the main station in the Netherlands. All freight trains from Utrecht (Ut) to Emmerick (Em) that usually run via Geldermalsen, are now rerouted via Arnhem (Ah).

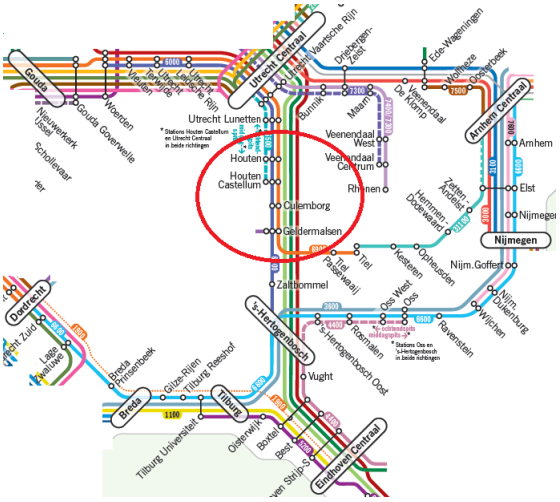


Figure 6: Constructions in case 2

7.1.3 Instance definition

If we want to find a solution for the two cases that we brought up, we need to make choices for the model parameters. For the feedback loop, we choose a bound on the deviations of event times, and we choose how many services we cancel in each iteration. Together with the case, these two choices define an instance.

As every event can be scheduled at T different times, the SAT solver uses $T + 1$ dichotomous variables to represent one integer valued variable ν_e in the model according to Constraints 4.7 to 4.28. Also, each PESP constraint requires at least T SAT lines to represent the same feasible region. It is not hard to imagine that the size of such a model can increase rapidly and could exceed the available memory of the computer. Thus, the parameter choices for an instance play an important role in the size of the model.

Also, it is not desirable to give the SAT solver full freedom to choose all possible times for the events, as it is the objective in TTAP to minimize deviations and cancellations. To minimize deviations and to limit the model size, we restrict the domain $[0, T-1]$ of the original event time ν_e to $[(\pi_e - maxDev) \mod T, \quad (\pi_e + maxDev) \mod T]$. The parameter $maxDev$ represents a maximum deviation from the original event time.

7.2 The SAT solver

The implementation of the SAT formulation of TTAP as proposed in Chapter 5 obtains results that we present in this section. We use $T = 600$ in all cases, such that an hour is divided in 600 time units that present tenths of minutes. The parameter $maxDev$ for the maximum number of time units that we allow an event time to differ from the original time, differs per case study. For all results, we report the used value for $maxDev$. As we use $T = 600$, the maximum value for $maxDev$ is 300, which means that we can choose at most a maximum deviation of 30 minutes.

The results are obtained on a desktop with an Intel® Xeon® core, a fourth generation E5-processor at 3.60 GHz and a RAM memory of 16 GB. The model was implemented in Java 17 in an Eclipse IDE for Java Developers 2022-06. We used the open-source SAT solver MiniSAT 1.14 that was introduced in Eén and Sörensson (2003). If we apply no construction works, set $T = 600$ and $maxDev = 0$, and use all PESP constraints, it takes about 3 seconds to build and solve the problem.

To evaluate solutions, we use the same objective function with the same weights as in Maróti and Vollebergh (2021). For passenger services, we apply a penalty of 100 for every part that we cancel. For a forward shift in event time, we use a penalty of 1 per minute, and a penalty of 2 per minute for a backward shift in time. For freight trains, we apply a penalty of only 0.01 per shifted minute, as delays are not a big problem for freight trains. Moreover, we do not allow freight trains and deadhead services to be cancelled. In all runs, we require fixed boundary times and include all PESP constraints.

7.2.1 Geographical impact

If we solve the model, we do not necessarily need to include all events that take place in the Netherlands. We only consider the region near the construction works. Also, for partial and full blockades, TTAP modelling encounters different challenges, which we also briefly discuss.

Consider a region A regular timetable for NS consists of 9,000 to 10,000 events per hour for passenger services and freight trains. Planned construction works do not affect all of these events. For services that do not run through or near construction works, NS mostly aims to maintain the original timetable. Therefore, it makes sense to only consider services running through and near an affected area. To make this possible, we truncate the timetable at a geographical shape around the affected area. Currently, we choose such a region by human insight and expertise, although it is possible to predefine areas in the Netherlands and just check in which area the construction works take place.

When considering a truncated timetable, it is important to realize that the events that are not considered still have influence on the events that are in the model. For instance, a service that runs through the affected area may also run outside of the affected area. Moreover, trains have to adhere to headway times for other trains that are completely outside the affected area. To make the adapted timetable in the modelling region consistent with the existing timetable that is left out of the model, we fix the event times at the borders of the shape on the map.

Partial blockades If a partial blockade applies, it mainly increases the number of headway PESP constraints. For instance, if one track is available instead of two, more trains run over the same track. This yields more combinations of trains that need to keep headway times in consideration. It may even be so that trains in opposite directions now make use of the same tracks, while that was not the case before.

For partial blockades, it is likely that the only solution is to cancel some services, as the track capacity is more limited than usual. It is not trivial which services, though, and there are many combinations of service cancellations that possibly lead to feasible outcomes. It is possible to reduce the number of combinations and outcomes by requiring the model to cancel pairs of services or series all at once. This may worsen the solution compared to cancelling services one by one, but reduces the number of possibilities and therefore may decrease the running time. We include the number of services that are cancelled per iteration of the Feedback Loop as a parameter.

Full blockades For full blockades, it is obvious that the services running at fully blocked tracks must be cancelled. We cancel these services between the nearest turning points compared to the tracks. When we split services like this, these trains mostly stay longer at stations of their new ending and starting events compared to their former dwelling activities there, occupying track capacity. Let us now review the results that we obtained.

7.2.2 Case 1 - general analysis

In Table 1, we present the results for case 1 that we obtained by using the SAT solver in the Feedback Loop algorithm. Each row represents the results of one instance. In the first two columns, we present the instance by showing the input to the model, namely the maximum deviation of event times in minutes and the number of services that we cancel per iteration. In the third column, we show how many services we have cancelled for this instance. Column four represents the number of iterations. In the fifth and sixth column, we present the objective value and the running time in seconds, respectively. Note that the number of iterations in the fourth column can easily be deducted by dividing the number of cancelled services by the number of services that we cancel in each iteration and adding one for the feasible final iteration.

Table 1: Results for case 1

max. deviation in minutes	# cancelled services per iteration	# cancelled services	# iterations	objective value	running time in seconds
2.5	1	4	5	8,349.59	2,093
5	1	3	4	15,409.73	4,393
5	2	6	4	14,295.69	5,013
10	1	6	7	25,682.82	5,491
10	2	8	5	23,945.66	3,380
10	series	50+	10+	29,271.27	1,621
20	1	3	4	44,014.44	9,078
20	2	6	4	40,183.66	6,889
none	1	3	4	48,797.71	22,671

Let us now discuss the computational results of case 1 that we presented in Table 1. If we restrict the deviation to 20 minutes and cancel two services per iteration, we cancel 6 services in total, and we also obtain a high objective value compared to runs in which we give the model less freedom. As more deviation is allowed and the model has no incentive to minimize deviation, it makes sense that event times deviate much more from their original time compared to cases where the deviation is constrained to e.g. 10 or 5 minutes. If we apply a maximum deviation of 10 minutes, we can see that the running time decreases if we cancel more services per iteration. This is not the case because we need fewer iterations to find a feasible solution. The number of necessary iterations even increases, as well as the number of cancelled services. However, as many constraints are deleted when a service is cancelled, the problem size decreases rapidly over the iterations. The SAT solver then takes less time to solve the problem and find a MUS.

It also stands out that with lower values for the parameter of the maximum deviation, we sometimes also cancel fewer services. For instance, when applying a maximum deviation of only 5 or even 2.5 minutes, we only cancel 3 and 4 respectively, while for a maximum deviation of 10 or 20 minutes, we cancel at least 6 services. At first sight, this pattern appears very strange. It would make sense if fewer services are cancelled if a model gets more freedom to shift events. However, this counterintuitive result is probably due to the iterative nature of our algorithm. Presumably,

the solver uses quite much of the freedom to shift event times to solve conflicts. Although these large shifts may cause more conflicts in the future and lead to more cancellations in later iterations, the solver does not incorporate this, and just keeps cancelling services and searching for feasible solutions until it finds one. This way, it could be that giving the model much freedom to shift event times later results in bigger issues so that more services are cancelled.

7.2.3 Case 1 – instance specific analysis

In Table 2, we present the results for the instance of case 1 with a maximum deviation of 20 minutes for event times and cancelling 1 service per iteration. The first column of the table represents in which iteration of the Feedback Loop we currently are. Then, in the second column, we can see how many constraints the MUS consisted of, followed by which service was cancelled in the particular iteration and the running time of the iteration, respectively in column three and four.

Table 2: Case 1 – maximum deviation of 20 minutes – cancel 1 service per iteration

iteration	# constraints in MUS	cancelled service	running time in seconds
1	2	A800	3125
2	4	B3600	3024
3	51	A6600	2921
4	0	none	8

Now, we dive deeper into the details of the results of some instances. First, we take a closer look at the results of the instance with a maximum deviation of 20 minutes and cancelling one service per iteration. The results are presented in Table 2. First, we encounter a conflict for service A800 at Geldermalsen. The conflict consists of only two PESP constraints and is probably caused by shifting the event times to fit the rerouted freight trains fit in. In the second iteration, we find conflicts between the intercity services A3600 and B3600 and the sprinter service C6600 at Ravenstein and Oss. That makes sense, as the construction works take place on those tracks. In the third iteration, we find 51 conflicts. These conflicts contain services H3100, D3600, A6600, D6600, and F7600, and also take place near Ravenstein. Service A6600 is present in almost all conflicting constraints, so that service is cancelled in this iteration. After that, the fourth iteration provides us with a feasible solution.

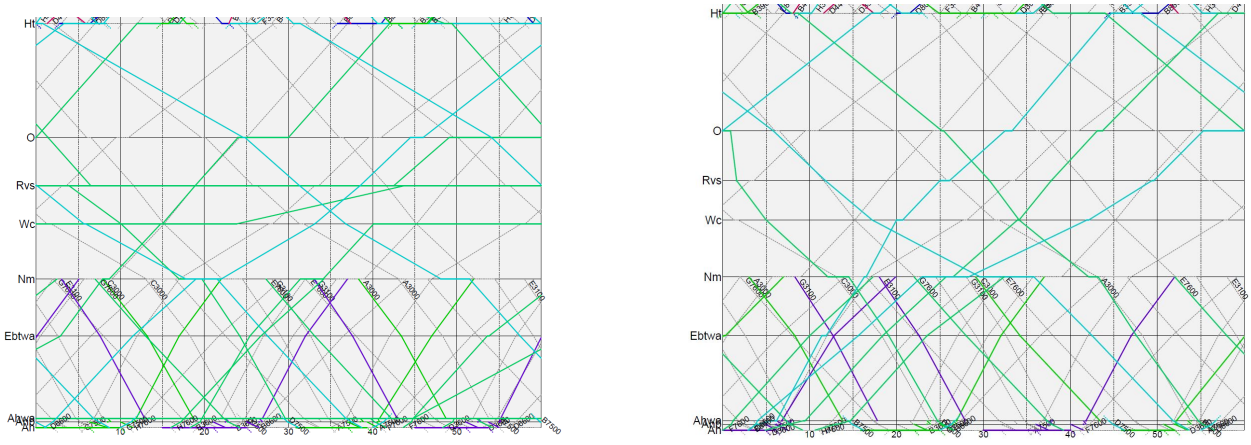
In Table 1, we can see that there is also another instance which also cancels the minimum number of services, namely 3. This is when we apply a maximum deviation of 5 minutes, and the objective value is even lower. We analyze the iterations of the Feedback Loop by presenting Table 3, which is of the same format as Table 2.

Table 3: Case 1 – maximum deviation of 5 minutes – cancel 1 service per iteration

iteration	# constraints in MUS	cancelled service	running time in seconds
1	2	H1100	730
2	4	B6600	852
3	49	A3600	1921
4	0	none	15

For this instance, we see very similar things happening compared to the instance with a maximum deviation of 20 minutes. One difference, however, is that we cancel a different service in the first iteration, namely the service H1100. Probably, there are multiple possibilities for the SAT solver to find a MUS, as multiple services do not fit in and multiple MUS options have the same size. Then, the intercity service and sprinter service get cancelled in a different order and also for different directions, but we do not detect any other significant changes.

Let us also take a look at the structure of these two solutions, which we present in so-called Time Space diagrams in Figure 7. We plot services with space on the vertical axis against time on the horizontal axis. Thus, at each point in time, we know where all services are, if we cut through the graph vertically. On the other hand, we know for each station which services are present at what times if we cut the graph horizontally at the particular station. In these figures, the grey lines represent the original timetable, and the coloured lines represent the new schedules for every series. All Time Space diagrams have enlarged versions in the Appendix.



(a) Maximum deviation of 5 minutes

(b) Maximum deviation of 20 minutes

Figure 7: Time space diagrams for the solutions of the Feedback Loop algorithm for case 1.

In Figure 7, we can see that for both instances several services have been cancelled between 's Hertogenbosch and Nijmegen. In Figure 7a, a maximum deviation of 5 minutes applies for event times. We can see that this results in situations where activities take a bit longer than before. The time that an activity takes can be deducted from the steepness of the corresponding lines. As a line is the connection between two event times, it appears steeper if the activity takes shorter. If we

schedule the event times such that we take longer for an activity, the line between the corresponding stations is flatter. This happens, for instance, with the light blue lines between 's Hertogenbosch and Oss, which are flatter than the corresponding grey lines.

Then, between Nijmegen and Arnhem, most services are shifted, so that we can see the grey lines as well as the coloured lines. It appears that most event times have been shifted proportionally to each other, such that activities roughly still take the same amount of time. We deduct this from the fact that the steepness of the grey lines corresponds to the coloured ones, approximately. Although we expected that a solution with a maximum deviation of 5 minutes would look better than one with a maximum deviation of 20 minutes and much more shifts, it appears that both solutions look similar. In both solutions there are quite some changes in event times, and there is not one that appears particularly better than the other. The objective values from the table, however, shows us that a solution with a maximum deviation of 5 minutes is better.

7.2.4 Case 2 - general analysis

In case 2, a full blockade applies for a part of the tracks on which construction works take place. We cancel the service parts that run via the blocked corridor in advance. We present the results in Table 4, which has the same format as Table 1.

Table 4: Results for case 2

max. deviation in minutes	# cancelled services per iteration	# cancelled services	# iterations	objective value	running time in seconds
2.5	1	5	6	19,175.08	2,362
5	1	2	3	17,642.32	4,393
5	2	4	3	24,173.68	5,013
10	1	2	3	29,414.52	1,447
10	2	4	3	27,917.37	1,442
10	series	50+	10+	32,782.53	1,499
20	1	1	2	47,760.46	745
none	1	1	2	47,955.46	774

We find that the algorithm needs less running time compared to case 1. This is probably partly due to cancelling parts of services in advance, which takes most of the pressure off the problem area. We also notice that solving the case with a maximum deviation of 5 minutes takes three times as much runtime as the other values for this parameter. Unfortunately, we were not able to find a reason for this behaviour. Moreover, we see roughly the same patterns as for case 1. If we provide the solver with more freedom, the objective value increases as well as the number of cancelled services.

For a maximum deviation of 20 minutes, however, the number of cancelled services is lower, namely one service. From the objective value, we deduct that event times are shifted quite much, though, as the objective is still higher than runs with more cancelled services. Also, the computation time is much lower, which we can explain by having to cancel only one service. We conclude that

it can be rewarding in terms of computation time to give the solver more room for deviating event times, but that it is likely that the objective value increases.

Another remark that we can make, based on Table 4, is that if we cancel two services per iteration instead of one, the number of cancelled services doubles. This means that cancelling these extra services does not solve or prevent any other conflicts. However, another interesting result that comes with it, is that the objective value decreases. This seems strange at first, but could be well explained. Namely, it could be that the events of the cancelled services were moved more than 100 minutes in total in the other solution. That would make the objective value higher and more expensive than cancelling the services. If a planner thinks this result is undesirable, it is easy to solve by changing the ratio between the weights in the objective function.

7.2.5 Case 2 – instance specific analysis

In Table 5, we present the results for the instance of case 2 with a maximum deviation of 10 minutes for event times and cancelling 2 services per iteration. This table is of the same format as Table 2, apart from the fact that multiple services are cancelled and presented in the third column.

Table 5: Case 2 – maximum deviation of 10 minutes – cancel 2 services per iteration

iteration	# constraints in MUS	cancelled service	running time in seconds
1	2	D800 & H3500	680
2	2	G3500 & D3900	752
3	0	none	3

We now consider the instance specific results, as presented in Table 5. In the third column we first mention the service that occurs in most conflicts. Then, the second service is a service that turns upon the first service and appears in most conflicts compared to all other turning services. However, in this instance we take a random turning service in both iterations, as no turning service appears in the MUS in either case.

In the first iteration, we have a conflict at the South side of the construction works, at the point where the line connects to the Betuweroute. This is a busy infra point, as many freight services pass by here. We cancel the intercity services D800 and H3500 to create more space for the other intercity services that run there. Then, in the second iteration, the conflict occurs in Geldermalsen, also at the South side of the construction works. This station is the new turning point for both intercities and sprinters than run between Utrecht en 's Hertogenbosch. To solve this, we again cancel two intercity services that run on the same tracks.

Let us also consider another interesting instance with a maximum deviation of 2.5 minutes and cancelling one service per iteration. As we have only very little room to deviate event times, the SAT solver needs to cancel more services. We present the results in Table 6.

Table 6: Case 2 – maximum deviation of 2.5 minutes – cancel 1 service per iteration

iteration	# constraints in MUS	cancelled service	running time in seconds
1	2	H6900	565
2	2	D3000	403
3	2	F8800	419
4	4	H3500	629
5	4	E6900	372
6	0	none	5

In Table 6 we present the results of case 2 with an applied maximum deviation of only 2.5 minutes and cancelling 1 service per iteration. In the first three iterations, we find a similar situation as before, namely a conflict of only two constraints. In these conflicts, a service no longer fits in the schedule because of rerouted freight trains. In these cases, the headway constraints mainly form problems. In iteration 4 and 5, more services appear in the conflicts, and running constraints as well as headway constraints show us that the tracks are too busy for the combination of services that we try to schedule there. In both iterations, one service is cancelled to provide enough room for the other services to be scheduled. Cancelling service H3500 in iteration 4 is unfortunately not enough to solve everything. Multiple services that appear in the conflict of iteration 4 are still present in that of iteration 5, so that we also need to cancel E6900.

7.2.6 General observations

In addition to the case-specific findings that we presented in the previous sections, there are some observations that hold for both cases. First, it appears that we always cancel many services if we cancel a whole series in each iteration. We conclude that we obtain better results if we cancel at most two services per conflict, as cancelling more services does not seem to prevent other conflicts from occurring. However, this may also be due to some implementation error on our side. Moreover, cancelling two services at once also does not work out well, mostly. In most cases, twice the number of services get cancelled, without much reward.

Secondly, we find that the model never chooses to plan an extra dwell. We included this option via Constraints (4.5) that we introduced in Section 4.1.2. Although it seemed nice to provide the solver with more opportunities to prevent conflicts, the SAT solver does not take advantage of this possibility. Apparently, there is not much benefit in it from a scheduling perspective. However, if all services through a particular station are cancelled, although it is still accessible, it might be worth including from a travelers' perspective. As we lack an objective function in SAT, it would be best to add PESP constraints that force the extra dwell. Then, if we observe such a constraint in the MUS, we could choose to throw them out first. We will further elaborate on this idea in Section 8.1 as it is a direction for further research.

Thirdly, we observe that calculating a MUS is a very time-consuming task for the SAT solver. In all cases, it takes at least 99% of the total computation time. In the Feedback Loop algorithm,

we use the MUS to decide which service(s) we cancel before we retry to solve the problem. In the instance specific analyses that we have discussed, we can see that the MUS often consists of only a few conflicting constraints.

We also tried to solve a case involving three construction regions. Unfortunately, we ran into memory overflow as a result of the big problem size. We have not been able to resolve these memory issues. Let us also examine the structure of our solutions somewhat further.

Deviations The first observation that we make is that deviations are applied to almost all event times, as we cannot penalize this in the SAT solver. Also, the maximum allowed deviation is applied to many event times. This is probably due to the solving strategy of the SAT solver that tries finding a solution by first setting all variables to false, and then only assigning a value true to the variables for which that is necessary. In the final solution, for many event times still all variables are false, so that the event time takes the value of its upper bound.

Moreover, we observe that it is more common to shift all events of a series in the same direction and roughly to the same extent, rather than shifting a few events to make it fit. We even detect situations in which similar services of the same series have been exchanged in the timetable. This is heavily penalized in the objective value, as all events of these services present a large deviation, while in practice not much has changed. On the other hand, we do not detect many swaps between different services. For instance, it barely happens that an intercity service is switched in time with a sprinter service, let alone multiple at once. This has the positive side effect that the frequency distribution of services is approximately preserved.

Activity durations Moreover, we also detect that the SAT solver makes structural changes to dwelling times and running times as well. We first explain how dwelling and running durations are scheduled in the original timetable, and then indicate how the SAT solution differs from this. All activity time spans have lower bounds and upper bounds. For driving activities, we want the time it takes to be close to the lower limit, so that we don't run too much below maximum speed. In the original timetable, a train almost always runs at maximum speed. If it is necessary to arrive later at a certain station, this is mostly taken care of by prolonging the dwelling time at previous stations. As these choices are embedded in the original timetable, it is not necessary to cover them explicitly in TTAP. Namely, trying to minimize deviations from the original timetable, also implies that we want to maintain these choices.

In the solution that is provided by SAT, however, dwelling lengths are reduced, and running times have extended. As we have no objective function and therefore do not include incentives to stay close to the original timetable, we lose these choices in the process and end up with a timetable that has quite different details. For the instances of case 1, we observe that respectively 67.2% and 58.9% of the dwelling activities lasts shorter in our solution than in the original timetable, and for the running activities 43.1% and 38.2% of the activities take longer. Then for case 2, we find that respectively 51.3% and 58.9% of the dwelling activities last shorter in our solution than in the original timetable, and 27.1% and 31.2% of the running activities take longer.

7.3 Comparison to NS implementation

The cases that we have used to analyze the performance of our model, are also used for testing Maróti and Vollebergh (2021) as well as by the groups of students that worked on this problem. We have rerun the results of Maróti and Vollebergh (2021) on the same machine that we used for our own model in order to conduct a fair comparison. As a baseline, we run the code without construction works to find out how long it takes to build the model. Without any construction works, building the RAAD model takes 108 seconds. This is way more than the 7 seconds of the SAT solver, but this makes sense, as this model includes more aspects of the problem.

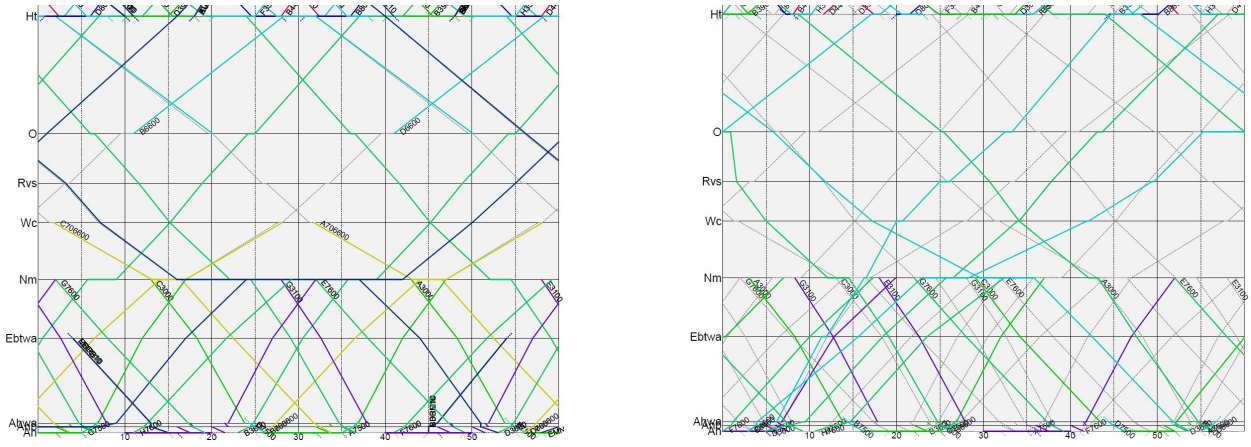
Because the findings have not yet been published, we offer the results that we get by running the MIP model as explained in Section 4.2 in Table 7. Maróti and Vollebergh (2021) use of some heuristics, which we mention without going into detail. They apply the heuristics in the following order: *Affected Area* (AA), *Bigger Affected Area* (BAA), *Affected Lines* (AL), *No heuristic* (none). The procedure is iterative, so each heuristic uses the final solution of the last one as a starting point.

Table 7: Results by Maróti and Vollebergh (2021)

case	heuristic	objective value	running time in seconds
1	AA	761.20	34
	BAA	761.20	13
	AL	761.20	1,498
	none	761.20	2,288
2	AA	15,760.64	45
	BAA	15,875.75	11
	AL	15,760.64	1,045
	none	15,760.64	1,033

In Table 7 we find the results of the model in Maróti and Vollebergh (2021). It stands out that CPLEX mostly finds the optimal solution already while using the AA heuristic. As the running time of the model with no heuristic only represents the last part of solving the problem, using the earlier runs, we add up all running times to get the total running time that it takes to get the solution. For case 1 the solver took 3,834 seconds in total. For case 2, it took 2,143 seconds and for case 3 1,100 seconds. It stands out that the CPLEX solver is both faster than the SAT solver, and it obtains better solutions as well. However, for planners this running time is still a bit too long. Especially if the results are not practically desirable and the algorithm has to be run several times with slightly different settings.

Let us also compare the two solvers in terms of solution structure. We present time-space diagrams for the CPLEX model in Maróti and Vollebergh (2021) and our Feedback Loop algorithm with the SAT solver. In Figure 8, we present these diagrams for the instance of case 1. For the SAT solver, we set a maximum deviation of event times of 20 minutes and cancelling one service per iteration.



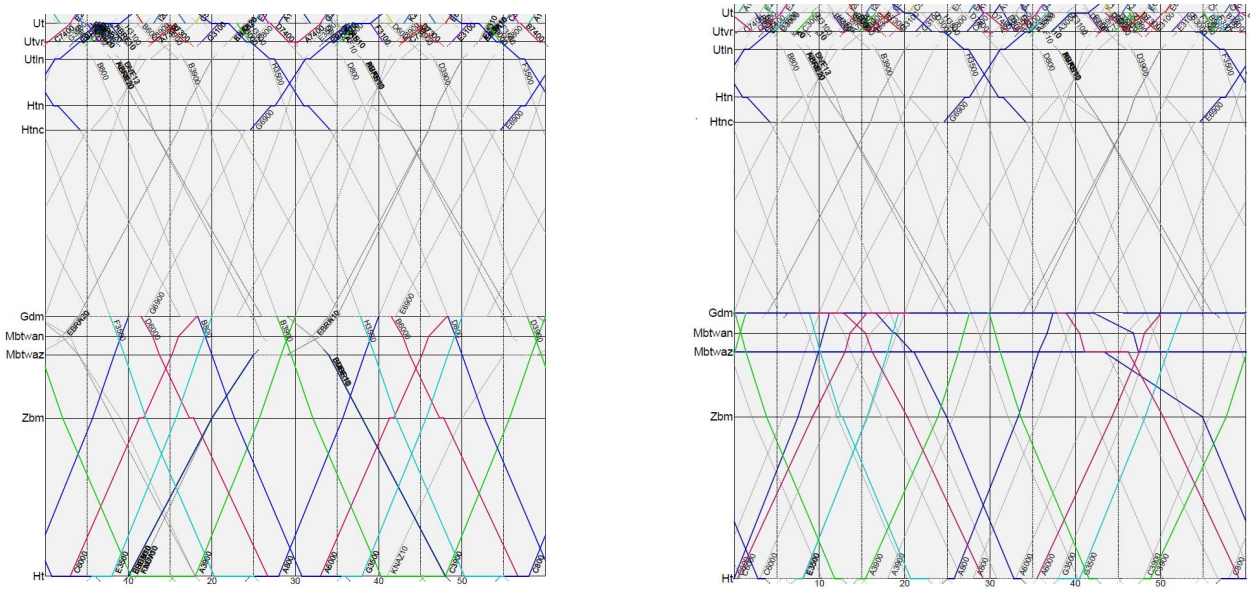
(a) RAAD model

(b) Feedback Loop algorithm

Figure 8: Time space diagrams for case 1

In Figure 8a we can see that the model has cancelled some lines between Oss (O) and Wijchen (Wc), as grey lines appear there. For the services that are still running, almost none is shifted in time. As these services present a grey and coloured line at the same spot, we only see the coloured lines. In Figure 8b, however, we can see that much more is going on. First, two more services are cancelled, that are presented by the dark blue line in Figure 8a. Also, between Nijmegen and Arhnem almost all services are shifted. Here, it is very visible that RAAD has an objective that prevents this from happening, and the SAT solver does not.

Let us also inspect such diagrams for case 2 in Figure 9. For the Feedback Loop algorithm, we choose the case with a maximum deviation of 10 minutes and cancelling 2 services per iteration.



(a) RAAD model

(b) Feedback Loop algorithm

Figure 9: Time space diagrams for case 2

In Figure 9, we see that the RAAD model and Feedback Loop algorithm come to similar solutions. Between Utrecht (Ut) and Houten Castellum (Htnc), the two solvers both choose to maintain services 6900 and 3500 up to Houten Castellum. Between Houten Castellum and Geldermalsen (Gdm) we only see the original timetable. All other services are cancelled there, as there are no tracks available there. Then, for the lower part of the figures, we see the same differences as in Figure 8. The Feedback Loop algorithm again shifts quite some services, whereas the RAAD model nicely preserves the schedule. Compared to Figure 8b, however, the shifts have turned out neater here.

From these comparisons, we can conclude that the solutions from the RAAD model are both neater and obtained with less computation time. Only for case 2, the SAT solver can find an objective value of 17,642.32 if we apply a maximum deviation of 5 minutes. This is relatively close to the objective value of 15,760.64 that RAAD finds, which boils down to an optimality gap of approximately 11.9 per cent.

8 Conclusion

In this thesis, we consider the Train Timetabling Adjustment Problem (TTAP) of adapting a cyclic timetable in case of construction works. We aim to deal with a limitation in availability of tracks that we know of in advance. The goal is to create a new timetable that deviates from the original timetable as little as possible, in terms of changing departure and arrival times and cancelling services. Recent research has tried applying MIP formulations to this problem, but unfortunately the results were not yet adequate for all test cases. In this thesis, we therefore employ a satisfiability approach, which in the past has produced promising outcomes for other scheduling problems.

SAT solvers are known for finding feasible solutions very fast. A SAT solver determines if a problem can be solved to satisfiability or not. Mostly, this takes only a couple of seconds, even for huge problem instances. The SAT solver, however, is not specialized in optimizing a problem and a regular SAT problem formulation does not contain an objective function. There are MaxSAT solvers available, but those take away a part of the advantages of the SAT solver and are usually very slow. In this thesis, we therefore came up with a heuristic algorithm that takes on the task to minimize the deviation from the original timetable. This algorithm provides problems to the SAT solver and decides which services we cancel, based on the MUS that the SAT solver calculates.

Applying our approach to two real-life cases, we find that the SAT solver can solve TTAP instances, but the presented solutions are significantly worse than those of the CPLEX solver that solves the MIP model in Maróti and Vollebergh (2021). Unfortunately, in most cases the SAT solver is also much slower than the CPLEX solver. SAT determines very fast whether the given problem has a feasible solution. If one exists, SAT also provides a solution very quickly. However, if SAT finds that the problem is unsatisfiable, it takes very long to decide why and produce the MUS. Sometimes, this even takes too much memory and cannot be solved. Especially for bigger instances with multiple construction works taking place at the same time, the SAT solver gets out of memory and cannot provide a good solution.

8.1 Further research

For future research, it would be beneficial to look into improving the process of calculating a MUS. Namely, most of the running time of our algorithm is used by the SAT solver while determining the MUS. Based on this MUS, we decide which trains we want to cancel. Recently, much research was done on SAT solvers, so probably there are more efficient SAT solvers available. It is most rewarding to focus on a SAT solver that can find a MUS as quickly as possible. The downside of applying a different SAT solver is that most of these solvers are developed in other programming languages than Java. It is possible to write the CNF file in one language and solve the problem in another, but it would be most elegant to translate the code into the same language as the SAT solver is coded in.

8.1.1 Combining the SAT solver with the CPLEX solver

Another idea is to include the SAT solver in the branch-and-cut algorithm of CPLEX, so that we bundle the strengths of these two solvers. SAT is specialized in fast checking whether a problem is feasible, but not in optimizing a problem. A MIP solver, on the other hand, is specialized in optimizing by applying a sophisticated branch-and-cut method with educated branching decisions. It solves both the primal and the dual problem, searching for the best lower and upper bound for each node. The solver finds an optimal solution if the lower bound is equal to the tightest upper bound so far. Unfortunately, it can take quite a while to prove a solution's optimality.

Our hypothesis is that we can increase the speed of the CPLEX solver by calling the SAT solver in every node to check for feasibility. As theoretically a SAT-formulation is more efficient in determining infeasibilities than a MIP-formulation, this may speed up the calculations by earlier pruning of branches in the branch-and-cut algorithm.

In each node, we can exploit the SAT solver by quickly determining whether a feasible solution even exists. If the SAT solver cannot find a satisfiable solution, we immediately conclude that we should close this line of inquiry, prune the branch, and move on to other nodes. If the SAT solver finds a satisfiable solution, this means that we have found some solution in the feasible space, leading to an upper bound. However, as the SAT solver performs no optimization, it must be noted that we have not necessarily found the tightest upper bound. Therefore, we still need CPLEX to solve the subproblem in such a case.

If SAT finds that the branching decisions up until the current node is a collection of unsatisfiable constraints, the branch can be pruned in CPLEX immediately, without waiting for the CPLEX solver to find this out by solving the MIP. To let the SAT solver check whether the problem is feasible, we only take into account the constraints that belong to the services that cannot be cancelled anymore. The solver has a substantial incentive to fix the choice not to cancel services, due to the included cancelling penalty in the objective function. Especially in branches in which multiple services must run, SAT could be of great use. It is possible that these branching decisions already make the model unsatisfiable, because they no longer allow these services to be cancelled.

If the constraints of services that can no longer be cancelled do not form conflicts, the full problem is also satisfiable, for instance by cancelling all other services. Namely, if a service is cancelled, we no longer have to deal with its constraints. Then, the MIP model only consists of constraints that belong to services that are not cancelled. However, if the subproblem with services that we can no longer cancel is not satisfiable, it is certainly not possible to find a feasible solution for the full problem. Including other constraints tightens the feasible region, or in the best case, leaves it as it was. Thus, any unsatisfiable subproblem results in unsatisfiability for the full problem.

All in all, we have proposed a SAT formulation for TTAP by using PESP constraints and introducing some other constraints as well. Although the SAT solver does not yet outperform other solution methods, the SAT solver is promising and could be used in different algorithms.

Proelium finitum est. Dominus adiutor meus fuit.

References

Abio, I., & Stuckey, P. J. (2014). Encoding linear constraints into sat. *International Conference on Principles and Practice of Constraint Programming*, 75–91.

Batelaan, M., Bijl, M., Luik, I., & Van der Wind, M. (2022). Ns case - alternative hourly pattern [Unpublished report for EUR course].

Bešinović, N., Goverde, R., Nicholson, G., & Steele, H. (2021). Review of railway timetabling developments: Planning phases, goals and future directions.

Bešinović, N., Widarno, B., & Goverde, R. M. (2020). Adjusting freight train paths to infrastructure possessions. *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, 1–6.

Cacchiani, V., Huisman, D., Kidd, M., Kroon, L., Toth, P., Veelenturf, L., & Wagenaar, J. (2014). An overview of recovery models and algorithms for real-time railway rescheduling. *Transportation Research Part B: Methodological*, *63*, 15–37.

Cook, S. A. (1971). The complexity of theorem-proving procedures. *Proceedings of the third annual ACM symposium on Theory of computing*, 151–158.

De Best, D., Van der Kaaden, M., Sterk, N., & Vaes, W. (2022). Adjusting a cyclic railway timetable for passenger and freight trains in case of construction works [Unpublished report for EUR course].

Doxopoulos, C., Elhorst, P., Veldkamp, N., & Yalınbaş, U. (2022). Alternative hourly planning due to maintenance and construction possessions [Unpublished report for EUR course].

Ebbinghaus, H.-D., Flum, J., & Thomas, W. (2013). *Mathematical logic*. Springer Science & Business Media.

Eén, N., & Sörensson, N. (2003). An extensible sat-solver. *International conference on theory and applications of satisfiability testing*, 502–518.

Gattermann, P., Großmann, P., Nachtigall, K., & Schöbel, A. (2016). Integrating passengers' routes in periodic timetabling: A sat approach. *16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2016)*.

Ghaemi, N., & Goverde, R. M. (2015). Review of railway disruption management practice and literature. *6th International conference on Railway Operations Modelling and Analysis, RailTokyo2015, Narashimo, Japan, March 23-26, 2015; Authors version.*

Großmann, P., Hölldobler, S., Manthey, N., Nachtigall, K., Opitz, J., & Steinke, P. (2012). Solving periodic event scheduling problems with sat. *International conference on industrial, engineering and other applications of applied intelligent systems*, 166–175.

Großmann, P. (2012). Polynomial reduction from pesp to sat, 166–175.

Huisman, D., Kroon, L. G., Lentink, R. M., & Vromans, M. J. (2005). Operations research in passenger railway transportation. *Statistica Neerlandica*, *59*(4), 467–497.

Kroon, L. G., & Peeters, L. W. (2003). A variable trip time model for cyclic railway timetabling. *Transportation Science*, *37*(2), 198–212.

Liebchen, C., & Möhring, R. H. (2007). The modeling power of the periodic event scheduling problem: Railway timetables—and beyond. *Algorithmic methods for railway optimization* (pp. 3–40). Springer.

Louwerse, I., & Huisman, D. (2014). Adjusting a railway timetable in case of partial or complete blockades. *European journal of operational research*, *235*(3), 583–593.

Maróti, G., & Vollebergh, M. (2021). Het raad model. *Nederlandse Spoorwegen*.

Mooij, J. (2006). Een model voor het aanpassen van de dienstregeling bij een nachtelijke buitendienststelling van het spoor.

Nachtigall, K. (1996). Cutting planes for a polyhedron associated with a periodic network. dlr interner bericht.

Odijk, M. A. (1994). Construction of periodic timetables. part i: A cutting plane algorithm. *Research report*, 94–61.

Peeters, L. (2003). *Cyclic railway timetable optimization* (Doctoral dissertation EPS-2003-022-LIS). Erasmus University Rotterdam.

Serafini, P., & Ukovich, W. (1989). A mathematical model for periodic scheduling problems. *SIAM Journal on Discrete Mathematics*, *2*(4), 550–581.

Stut, W. (2009). *Een stochastisch optimalisatie model voor een robuuste dienstregeling: Een nieuwe oplos-methode* (Master's thesis). University of Twente.

Tamura, N., Taga, A., Kitagawa, S., & Banbara, M. (2009). Compiling finite linear csp into sat. *Constraints*, *14*(2), 254–272.

Van Aken, S., Bešinović, N., & Goverde, R. M. (2017a). Designing alternative railway timetables under infrastructure maintenance possessions. *Transportation Research Part B: Methodological*, *98*, 224–238.

Van Aken, S., Bešinović, N., & Goverde, R. M. (2017b). Solving large-scale train timetable adjustment problems under infrastructure maintenance possessions. *Journal of rail transport planning & management*, *7*(3), 141–156.

Van der Knaap, R. (2021). Cyclic railway timetabling using an iterative sat approach with a feedback mechanism.

Van Doorn, M., Willemse, V., Zhu, D., & Desmense, J. (2022). Creating alternative railway timetables for planned maintenance [Unpublished report for EUR course].

Vansteenwegen, P., Dewilde, T., Burggraeve, S., & Cattrysse, D. (2016). An iterative approach for reducing the impact of infrastructure maintenance on the performance of railway systems. *European Journal of Operational Research*, *252*(1), 39–53.

Veelenturf, L. P., Kidd, M. P., Cacchiani, V., Kroon, L. G., & Toth, P. (2016). A railway timetable rescheduling approach for handling large-scale disruptions. *Transportation Science*, *50*(3), 841–862.

Veenhof, H., Van Renssen, K., Koot, R., & Upis, R. (2022). Creating a cyclic disposition railway timetable in case of planned construction works [Unpublished report for EUR course].

Vollebergh, M. (2020). Incorporating flexible track use in the sat model of the dutch railway timetabling problem.

Wüst, R., Bütikofer, S., Ess, S., Gomez, C., Steiner, A., Laumanns, M., & Szabo, J. (2019). Improvement of maintenance timetable stability based on iteratively assigning event flexibility in fpesp. *8th International Conference on Railway Operations Modelling and Analysis, Norrköping, Sweden, 17-20 June 2019*, 1160–1177.

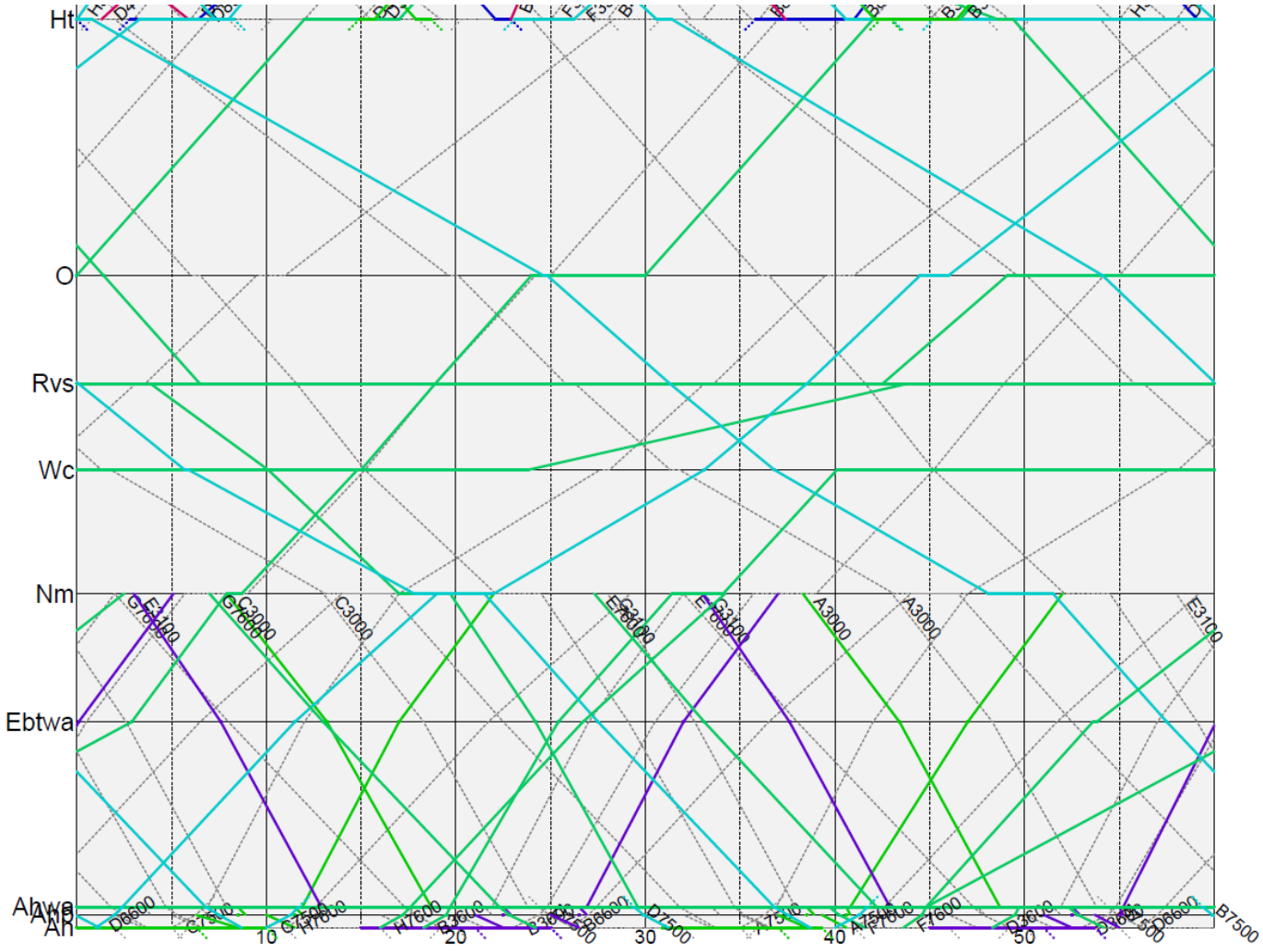
Zhan, H., Li, S., Wang, Y., Yang, L., & Gao, Z. (2021). Collaborative real-time optimization strategy for train rescheduling and track emergency maintenance of high-speed railway: A lagrangian relaxation-based decomposition algorithm. *Omega*, *102*, 102371.

Zhang, X., & Nie, L. (2016). Integrating capacity analysis with high-speed railway timetabling: A minimum cycle time calculation model with flexible overtaking constraints and intelligent enumeration. *Transportation Research Part C: Emerging Technologies*, *68*, 509–531.

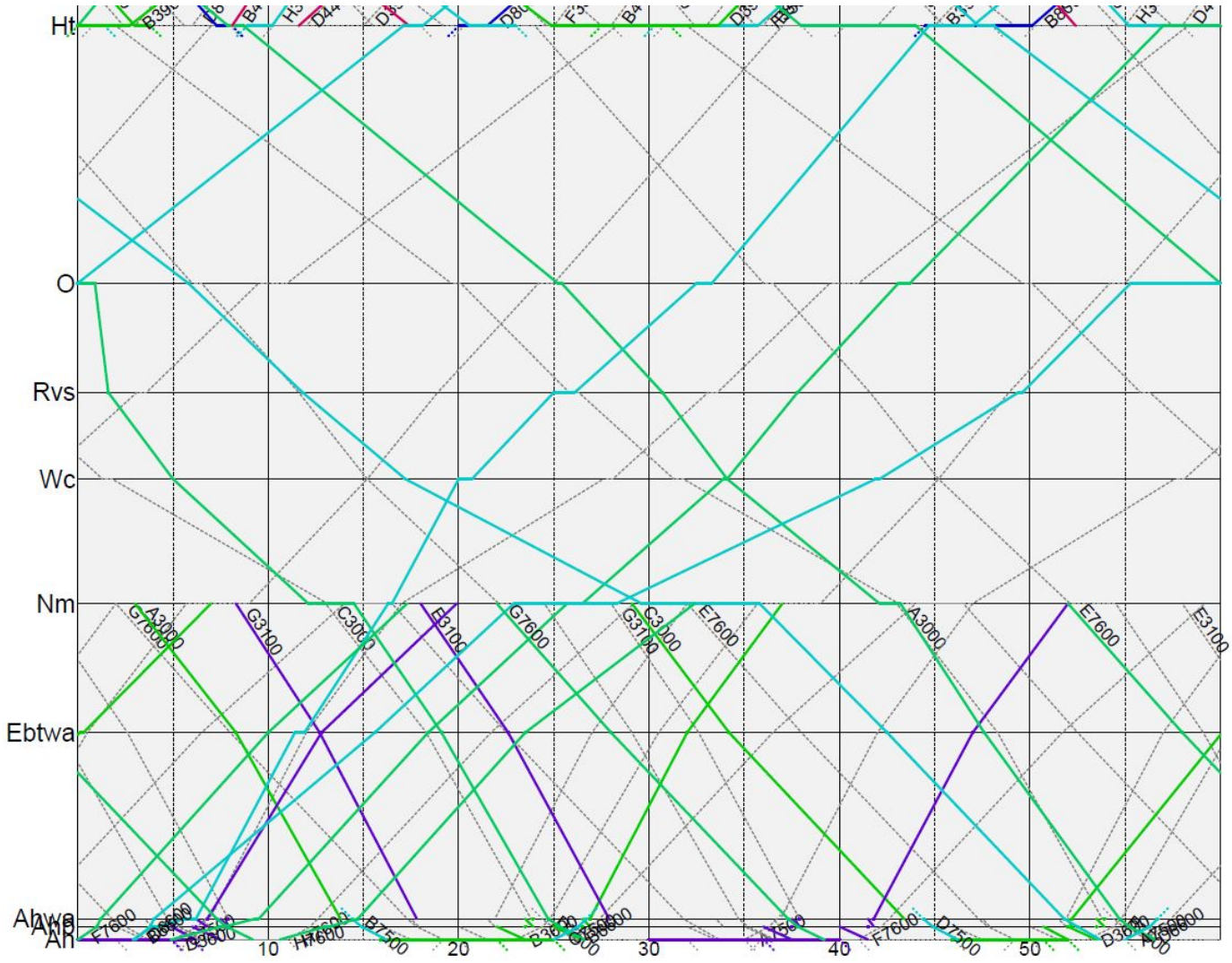
Zhu, Y., & Goverde, R. M. (2019). Railway timetable rescheduling with flexible stopping and flexible short-turning during disruptions. *Transportation Research Part B: Methodological*, *123*, 149–181.

Appendix

Enlarged versions of Figure 7

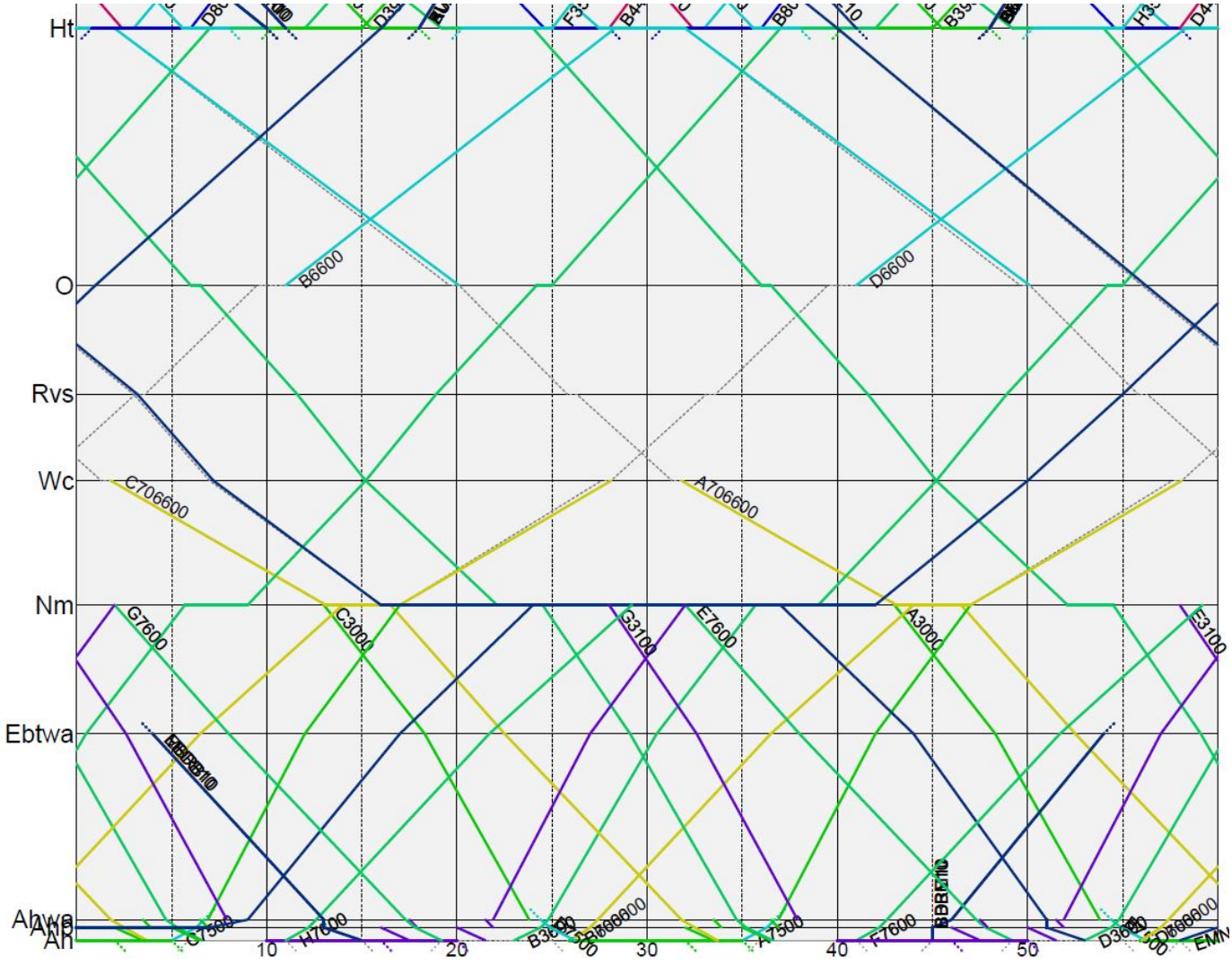


Maximum deviation of 5 minutes

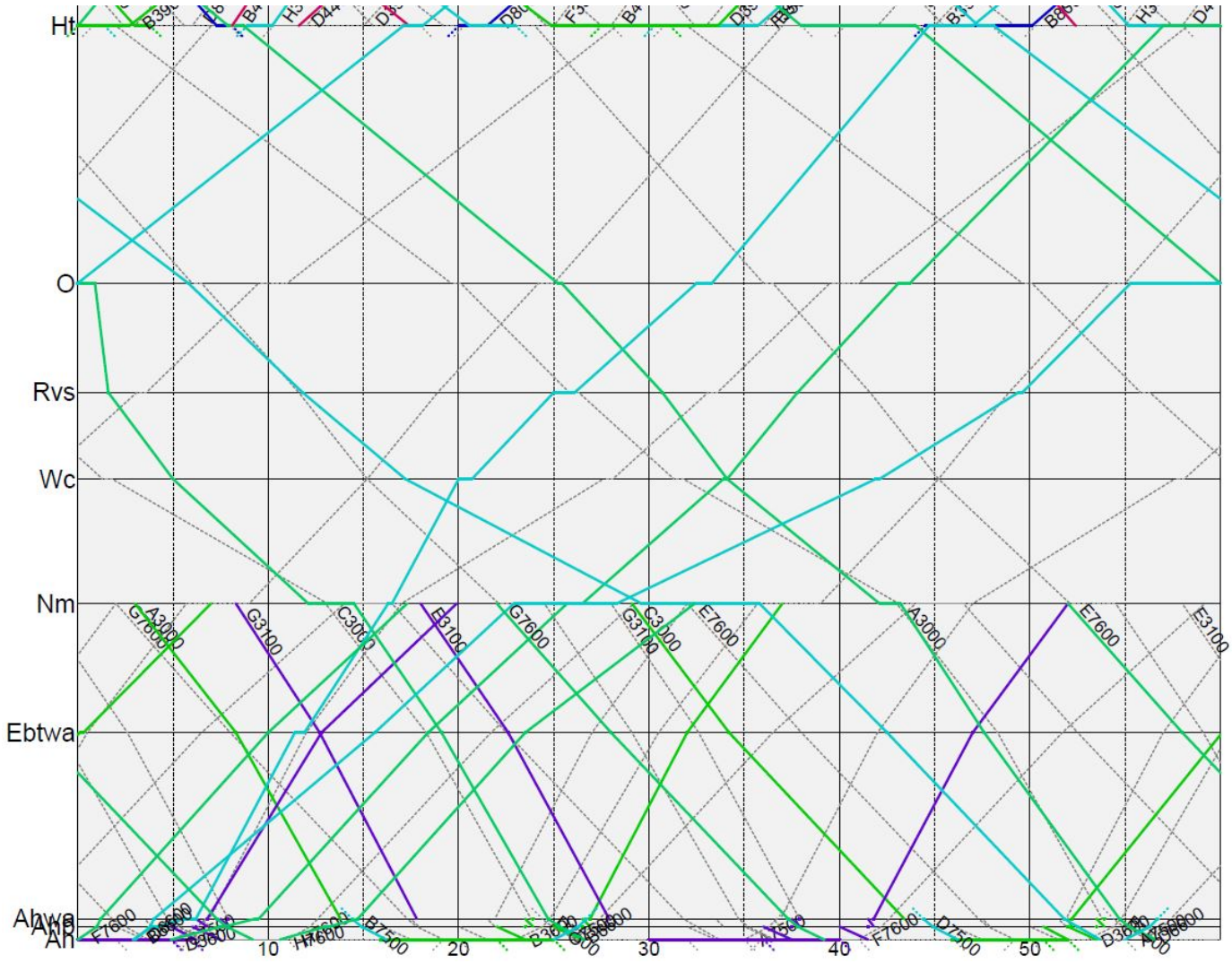


Maximum deviation of 20 minutes

Enlarged versions of Figure 8

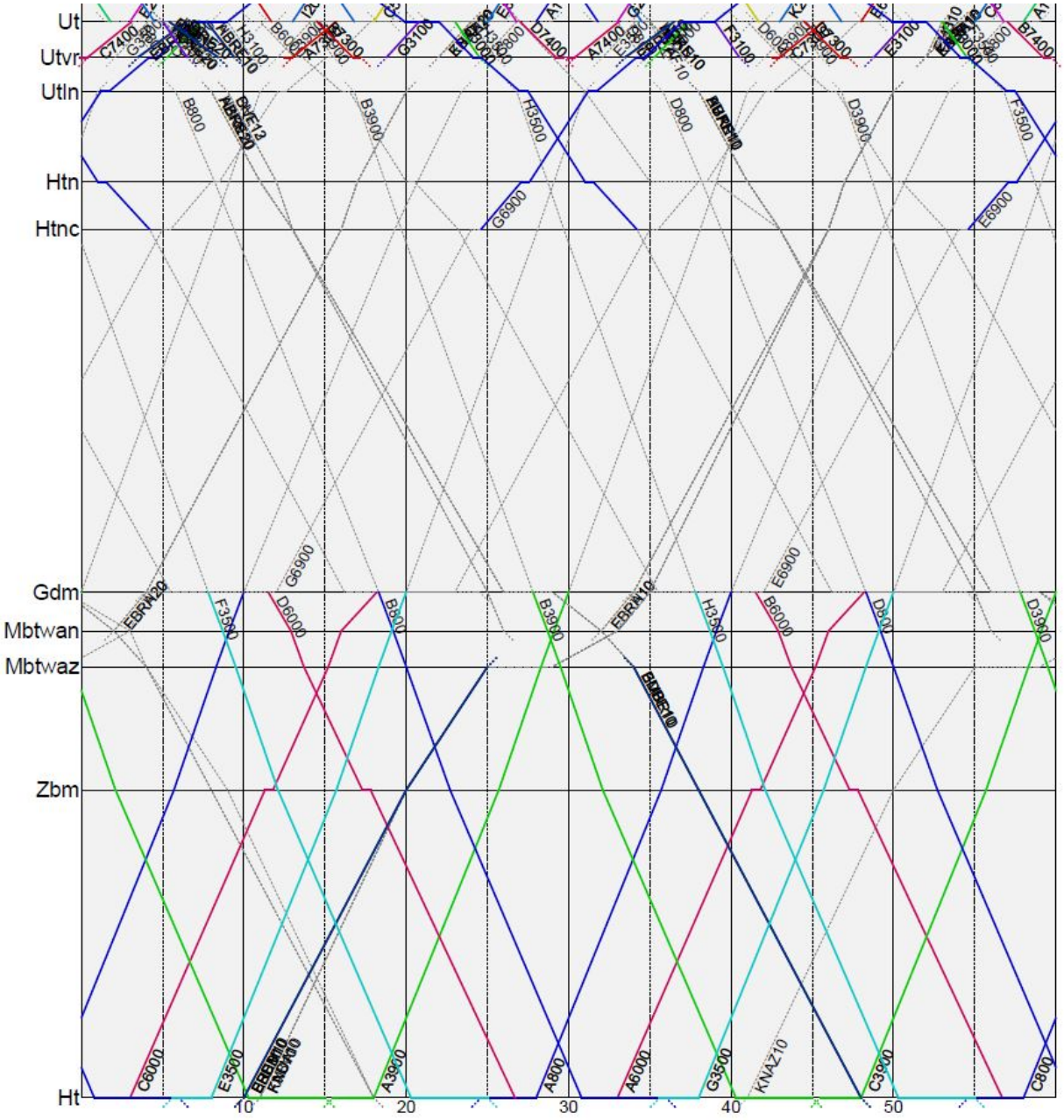


RAAD model

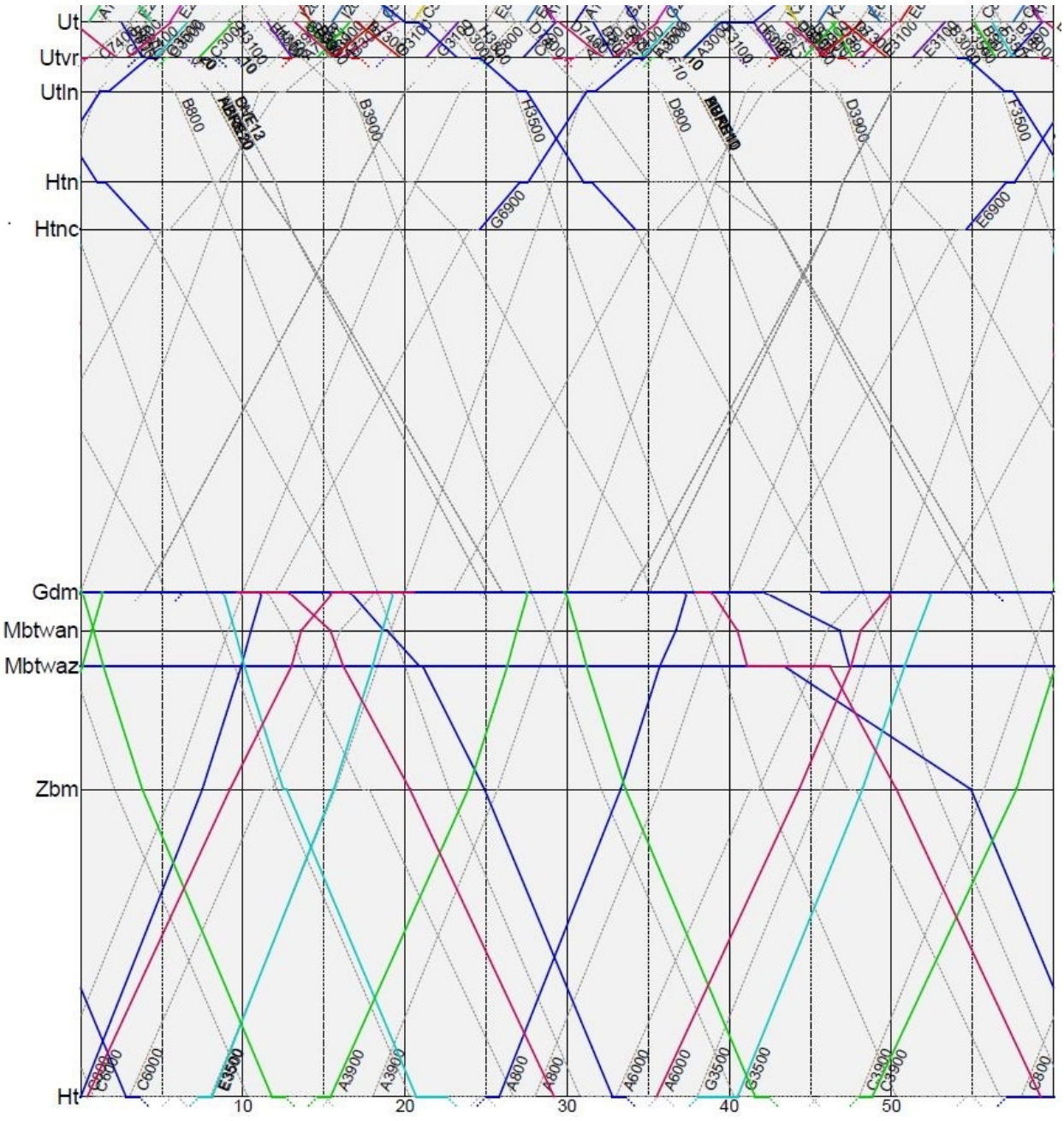


Feedback Loop algorithm

Enlarged versions of Figure 9



RAAD model



Feedback Loop algorithm