Creating New Train Timetables in Case of Disruptions

Optimising a Branch & Bound Algorithm

MSc. Thesis

Yoshi van den Akker

Creating New Train Timetables in Case of Disruptions

Optimising a Branch & Bound Algorithm

by

Yoshi van den Akker

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Thursday May 16, 2024 at 10:45.

Student number:	4594193	
Project duration:	September 2023 – May 2024	
Thesis committee:	Dr. N. Yorke-Smith,	TU Delft, supervisor
	Dr. Z. Erkin,	TU Delft
	A. van Schie, MSc,	CGI

Preface

Back in September last year, I started doing research for this thesis without a clear idea of what to expect for the next nine months. Optimising the algorithm of the VGB Solver, that I did not know at all at that point, would prove to be a challenging project. The experience I had with working on software related to rail infrastructure turned out to be very useful, but figuring out the details of this extensive codebase was a tough task nonetheless. Besides the valuable research experience this project brought me, I also learnt a lot about working on a large-scale software project. I got a taste of the working life through the internship at CGI, which prepared me for a good start of my career. Working on automating part of the incident handling process aligned perfectly with my goal for the future: building software that helps solve problems relevant to society.

Writing this thesis would not have been possible without the help of many people, whom I would like to thank here.

First, I would like to thank my TU Delft supervisor, Neil Yorke-Smith, for the many discussions we had on how to turn the project into a proper academic thesis. Then my gratitude goes out to the development team at CGI for helping me with understanding the details of the VGB Solver. From this team, special thanks go to Sven Kardol for his supervision, and Arjen van Schie for his critical feedback on the experiments I came up with. My thank also goes out to Wilco Tielman from ProRail, for helping me with obtaining data and for always being open to a discussion about new ideas. I am also grateful for the support of all other colleagues at CGI, for the occasional content-related conversations, but mostly for the casual chats at the coffee machines and the fun activities after work hours.

Last but definitely not least, I would like to thank my friends and family for their never-ending support. They put up with my complaints about how the experiments did not turn out the way I expected, or how code (that I wrote myself) did not do what it was supposed to do. Without their “gezelligheid” and ability to take my mind off of my thesis every now and again, I would not have been able to bring this project to a good end.

Yoshi van den Akker
Delft, May 2024

Abstract

The Dutch railway system is one of the most densely used systems worldwide and the busiest in Europe. Given the tight schedules, incidents can quickly cascade through the entire country if not handled properly. Alternative timetables are created to help train traffic controllers swiftly resolve such incidents. These schedules are currently created manually, but the team cannot keep up with demand. This is why CGI is developing VGB Solver, an application to automatically generate such timetables. The solver uses a branch and bound algorithm in which nodes are processed in best-first order, based on a heuristic value. For this thesis, different performance optimisations for this algorithm were implemented and analysed.

Various alternative formulations of the heuristic value, used to determine the likeliness of a node leading to a good solution, showed promising results. One of these formulas resulted in solutions for 96% more scenarios than before, and improved the quality of solutions for other scenarios.

Using machine learning, a decision tree was created to predict whether applying another new formula for the heuristic value gives better results than the old formula. This classifier achieved an accuracy of 73.5% on the test data. Having the solver choose between the old and new formula based on that classification resulted in some scenarios with worse scores, but twice as many improved, and the average improvement was higher than the average deterioration.

Recommendations are made to conduct further experiments related to the heuristic value calculation. Furthermore, it is suggested to separate the scores for evaluating solution quality from the heuristic value formula, to facilitate more fine-grained changes to the calculation of the heuristic value.

Contents

1 Introduction

The Dutch railway system is one of the most densely used systems worldwide and the busiest in Europe [1]. The infrastructure is managed by ProRail, which communicates the available capacity to the railway operators. ProRail's traffic controllers deal with over 4.500 incidents every year [2]. Such incidents can range from a tree falling on the track to a train not being able to move anymore, and from a switch being stuck to an accident involving road traffic at a level crossing. Sometimes train traffic must be stopped completely on the affected section of track, but sometimes parallel tracks may still be usable. Rail traffic controllers have to ensure the area around the incident remains safe, while aiming for train operators to operate the train services to the best possible level.

For more than 3500 pre-determined scenarios, alternative timetables (also known as VSMs, from the Dutch term "versperringsmaatregel", which roughly translates to "blockage measure") are available to help controllers reroute trains affected by an incident. Each of these VSMs describes how to effectively reschedule and/or reroute trains in case a specific part of the infrastructure becomes unusable. In 80% of incidents that involve obstructions on the tracks, one of these VSMs is applicable. Currently, the traffic controllers depend on the scenarios to handle the incident, which shortens the time until a stable alternative schedule is in effect. Railway companies rely on VSMs as well: firstly, to update their personnel and vehicle schedules during incidents, and secondly, to provide their passengers with alternative travel information quickly.

Currently, all VSMs are created by hand. The people who create these doing this are called "scenario makers". Some former rail traffic controllers are part of this group, as they have the experience to judge if the plans are feasible in practice. The demand for new VSMs has grown from 1100 in 2018 to 4000 in 2023. Currently, scenario makers cannot keep up with demand: 1360 of the 4000 requested VSMs for 2023 could not be delivered in time, and for 2024 it is projected that 800 VSMs will not be delivered at all. Given the tense Dutch labour market and the required domain specific knowledge for this position, finding new scenario makers is a challenge.

Automating the process of creating these scenarios is a logical next step. A team of engineers from CGI is working on the project titled "VGB Solver", in which VGB stands for "vooraf gedefinieerde bijsturing" (predetermined adjustment). Creating an algorithm to solve the problem exactly is possible, but probably not the best solution: some parts of the infrastructure allow for an almost infinite number of combinations of adjustments for all trains. Calculating all possible solutions may require weeks of computation time. For that reason, the team is working on a heuristic-based solution.

1.1. Scope

For the VGB Solver project, and therefore for this thesis, several assumptions and decisions are made to define the scope. Firstly, the solver is designed to work in situations for which all train movements are known beforehand, so a static environment. This implies scenarios will always be built on the assumption that all scheduled trains are on time and will not consider other disruptions than the one that scenario is made for. Scenarios should therefore not be seen as an exact solution, but rather as a guideline. If there are additional circumstances to be taken into account, the train traffic controller should make some adaptations to the adjustments suggested in the scenario.

The timetable for trains in the Netherlands is scheduled with an hourly pattern for the convenience of passengers. As the trains affected by an incident will be the same every hour, a scenario only has to describe adjustments to trains for one hour. These adjustments can be reused for the next hour(s) if the incident lasts longer.

Different timetables exist, for example those for peak hours and off-peak hours. Scenarios are made for both of these timetables. The third timetable, used during the night, is not in the scope of the project: as there are far fewer trains on the tracks during the night, adjusting their schedule during an incident can easily be done on the fly by the traffic controller.

A consequence of using the hourly pattern is that several types of trains are not taken into account for rescheduling. Cargo trains do not have a regular hourly pattern, so they cannot be added to the scenarios. The same holds for international trains: even though they have a regular service schedule, this is not based on an hourly pattern. If these trains are affected by the incident, the traffic controller must resolve this manually.

1.2. Research question

The problem of rescheduling trains in the event of incidents is so complex that the exact solution cannot be found efficiently. Therefore, the research question of this thesis is as follows: how can the solution quality of the VGB Solver be improved? This question will be answered using the following sub-questions:

1. How can the performance of the VGB Solver be improved by leveraging the existing optimisations better?
2. How can data about specific problem instances be used in the VGB Solver?

1.3. Thesis outline

The remainder of this thesis is structured as follows. Chapter 2 reviews existing research related to the problem of (re)scheduling vehicles and solving complex problems. Next, chapter 3 contains background information about how the existing algorithm works, and analyses the optimisation techniques that are implemented. Chapter 4 discusses experiments with changes to the algorithm and their effects. The conclusions of the thesis can be found in chapter 5.

2 Literature Review

This chapter presents a review of literature relevant to the research question from the previous chapter. First, an overview is given of research in the domain of public transport scheduling. After that, different methods to model (NP-)hard problems are reviewed.

2.1. Public transport (re)scheduling

Li, Mirchandani, and Borenstein [3] introduce the vehicle rescheduling problem (VRSP) in their work. They aim to minimise passenger delays caused by a bus breaking down by either rerouting buses nearby to the incident, or assigning a new bus from the depot as replacement. The vital difference from the problem for this thesis lies in the fact that the broken bus does not impact other vehicles (except for the replacement bus): any other bus can simply drive around the broken one. With incidents on railway tracks, the main problem is to find new routes without using the section(s) blocked by the incident. Furthermore, an incident on the tracks may not even involve a train that needs to be replaced, a fallen tree branch for example can also cause such disruptions. Despite the shared domain of public transport and the common problem of rescheduling, the work tackles a fundamentally different problem. To find more relevant literature, the search narrows to rail infrastructure specifically, as it is to be expected that literature in other subdomains of public transport will not tackle a similar problem.

Acuna-Agost et al. [4] provide a mathematical model to compute the optimal way of dealing with delayed trains, by potentially rerouting other trains to other tracks and/or platforms. Where this work differs from the requirements for the problem of this thesis, is that this model cannot deal with blocked tracks, only with delayed trains. Similarly, Caimi et al. [5] deal with the removal of conflicts from a train schedule, which is again meant for the regular timetable. These models are not designed to deal with track closures, which is the core of the research questions in this thesis.

Liu, Zhu, and Kang [6] tackle the problem of reallocating tracks within a station given one or multiple track closures. They do not use an exact mathematical programming solution, but instead opt for a genetic algorithm. For this solution to work, they need to make several assumptions, one of which is that the arrival and departure times of trains are fixed according to the timetable. With this limitation, some scenarios may no longer have feasible solutions, whereas they could work if trains could be delayed. Delaying trains is often used in practice and as it cannot be added to the model presented in this paper easily, therefore a different model should be used.

To model the railway infrastructure for problem-solving, different levels of detail can be used. Generally, they are classified into two categories: macroscopic or microscopic. “Macroscopic models consider stations as nodes and tracks as arcs between them, with given capacities, while microscopic models incorporate details such as block sections and corresponding signalling constraints” [7].

The work of Aken, Bešinović, and Goverde [7] uses a model of a macroscopic level, which means the details of the infrastructure may not always coincide with the solution found by their algorithm. Even though they add more real life constraints as an extension to the aforementioned model, “a microscopic counterpart, which can find feasible routes in station areas, has to be developed" [8]. Furthermore, partial blockages on open-track are not supported by this model, even though they are one of the major parts of the problem tackled in this thesis.

Another macroscopic model is presented by Louwerse and Huisman [9]. Their solution works for both complete and partial blockages, but it makes some simplifications. For example, for partial blockages they only consider the options of cancelling a train series altogether, or having it continue as planned (potentially with some delay). In real life, however, it would also be possible to have a train complete part of the designated route and turn around at a station before the incident (this is known as short turning).

Ghaemi, Cats, and Goverde [10] did use a microscopic model, but this only supports disruptions in the form of complete blockages. This significantly simplifies the problem, as rerouting to a parallel track is no longer a viable option. For that reason, all trains have to be short-turned at some station or cancelled. The only questions that remain are which station is the optimal location to short-turn at, and which trains to cancel. That is a much simpler problem, and it is only part of the problem this thesis is about.

The problem described and solved by Looij [11] is closest to the problem for this thesis. The author created a microscopic routing model for station areas to assist a macroscopic model for scheduling trains. Despite the microscopic level used in this work, it does not solve the same problem due to some restrictions. The main issue is that the model is tailored to station areas and "should be run independently for each complex station area" [11]. Incidents may occur outside the station areas, and the model should be able to find routes for such problems. Furthermore, this model requires a list of all possible routes to and from each platform and each entry/exit track of the station area as input. Creating such a list becomes infeasible if the areas in scope are no longer limited to only stations, as it would require an exhaustive search for all alternative routes for any area in the country.

2.2. Modelling/solving techniques

Most of the models presented in the literature in the previous section use some form of mathematical programming, such as mixed integer programming. Such models are based on a fixed number of variables, a list of constraints for those, and an objective function. As the problem of rescheduling trains involves many potential conflicts, the list of constraints can become extremely large. An example of such potential conflicts: for every routing point, e.g. a switch or a signal, two trains must be a certain period of time apart. This time may depend on the type of train, the type of routing point, and even more details. To capture the entire problem, the list of constraints would have a severe impact on the runtime of the model. Furthermore, during the development of the model, the exact conditions can change, also known as a "moving target" [12]. Modifying this model to accommodate such requests could present challenges, particularly when there are specific scenarios that require attention. Capturing these in mathematical constraints is not only a tough task, but also leads to an even more complex model. A significant amount of research has gone into using mathematical programming for solving the problem at a microscopic level, but little to no progress has been made after the works presented in the previous section.

Kuroiwa and Beck [13] describe a generic method of solving problems using dynamic programming. To solve a problem using dynamic programming, "the scoring system must allow the optimal solution to be broken up into independent parts" [14]. The work of Van Heuven Van Staereling [15] tries to solve the periodic event scheduling problem (PESP) using dynamic programming by decomposing the problem into trees. Unfortunately, the performance of that implementation was worse than other known solutions, and it could not solve all data sets. The problem posed for this thesis is even more complex: "PESP model is a pure scheduling model and not a routing model" [16]. Rerouting trains may directly impact other sub-problems, which means these cannot be solved independently. For that reason and based on the results in [15], dynamic programming does not seem to provide a feasible method of solving the problem.

The work of Eele and Richards [17] introduces a branch-and-bound algorithm for a collision avoidance problem. A major advantage of the branch-and-bound approach is the insight it gives into the solving process: by reviewing the decisions taken in finding the solution (and the rest of the decision tree), the behaviour of the algorithm can be analysed. However, computing the entire search tree still requires traversing the whole solution space, which may prove infeasible.

The previously mentioned methods would provide an exact solution, but given the NP-hardness of the problem, they are unable to accomplish this within a reasonable timeframe. A different, more common approach "is to relax the requirement of finding an optimal solution, and instead settle for a solution that is "good enough"" [18]. Such probabilistic algorithms can, for example, use generalisations or shortcuts to find satisfactory but (potentially) suboptimal solutions. These algorithms are known as (meta)heuristic algorithms. As a subtype of probabilistic algorithms, approximation algorithms define a performance guarantee, also known as the approximation ratio. This means that they will provide a solution that is not worse than x times the optimal value. Probabilistic search can also be combined with the aforementioned strategies. For example, heuristics can be introduced into a branch-and-bound solution to reduce the search space [17]. The choices for such heuristics are tailored to that specific problem, for example choosing the first obstacle encountered on the path or the one path that is least diverted. Other works on optimisations using branch and bound ([19], [20], [21]) also use specific heuristics for their problems. The last of the mentioned works, that of Tamannaei and Irandoost [21], also applies a more generally usable metaheuristic method: beam search. This limits the search tree of their implemented branch and bound algorithm using a heuristic selection method to obtain a result significantly faster.

2.3. Summary

A substantial amount of research has been done regarding solving (re)scheduling problems for trains. Most of the works use some form of mathematical programming to solve the problem, but do not use a model with enough detail to be used to replace hand-made scenarios. The work of Looij [11] stands out for the level of detail used and the similarity in the problems. However, the model itself is not usable directly due to a cut-off in the used infrastructure.

Furthermore, no literature was found that takes into account the workload of railway traffic controllers for rescheduling trains, whereas they are one of the stakeholders in the problem of this thesis. This perspective, along with the level of detail of the infrastructure this problem has to deal with, makes the project unique.

Applying heuristics to (otherwise) exact solution methods seems promising, but no research was found in which this is applied to (railway) scheduling problems. As such, this strategy will be one of the core points tackled in this thesis.

3 Background information on VGB Solver

This chapter describes the ideas and implementation used in the project, to provide the background information required to understand the experiments in the next chapter. The information is divided as follows. First, different algorithms are explained to solve the problem of creating alternative timetables. The following section describes the structure of the search tree used internally by the solver. Then, it is explained how the results from the algorithm are analysed. Lastly, several optimisation techniques that are applied in the solver algorithm are analysed.

3.1. Algorithmic options

The main goal of the VGB Solver project is to automate the creation of alternative timetables, as described in chapter 1. To solve this complex problem, various algorithmic approaches could be used.

Some form of artificial intelligence, for example reinforcement learning, could be applied. For supervised learning, the scenario makers could provide a way to give feedback on the output of the algorithm. However, a major disadvantage is that such techniques do not always lead to explainable results and may be hard to tweak in case of undesirable outcomes. Furthermore, a minor change in the scoring system could require retraining the model entirely. In the survey by Liebchen and Schülldorf [12] on optimisation projects for railway companies, 57% of optimisation experts reported experiencing a "moving target", a goal that is modified during the project. As such, being able to adapt internal details is an important requirement. Another key aspect pointed out in that same survey is the importance of complete input data. This aspect could also be a problem, as scenario makers often judge solutions based on their experience, and therefore no exact set of rules is known at the start of the project. In case specific adaptations have to be made, an explicit algorithm is likely to be a better option than a machine learning model.

Most relevant research works (see chapter 2) use a mathematical optimisation technique, for example (integer) linear programming. Besides how these models lack the precision to make specific adjustments (or require many constraints in order to do so) and the explainability criterion explained in the previous section, there is another reason to deviate from this approach. The existing literature has focused on this technique for a long time and has not made significant progress on microscopic scale recently, which means that to this date, no usable model has been created with the level of detail required for railway operations. Louwerse and Huisman [9] aimed to solve the problem in real-time and their work ends with the same conclusion: "for more complex cases or when we extend the model the computations times can increase considerably" [9]. In their conclusion, they recommend looking at different solution methods.

The VGB Solver is implemented with a branch and bound algorithm. Using the rules for traversing the search tree, the solving process is explainable and can be made comparable to that of human scenario makers. This way adjustments to the decision-making process can be made more easily than with machine learning or purely mathematical models.

3.2. Search tree

Branch and bound algorithms operate on a state space tree. Each node in the tree contains a part of the solution. The full solution can be found by backtracking to the root of the tree.

In the VGB Solver, each node stores the conflicts yet to be solved. The most important conflict is selected (see below for a further explanation), and all possible solutions to it are computed. The resulting adjustments form new branches that lead to children of the current node. This process repeats: processing a node results in adding children to it, each of which solves the most important conflict in the parent node in a different way. In this way, each layer of nodes in the tree solves one conflict, but in doing so, more new conflicts may be created.

3.2.1. Conflict order

To determine which conflict is the most important, a fixed order is used. This order is based on the difficulty of each type of conflict to solve, with the easiest problems being tackled first. The conflict of a route through the obstructed area has the lowest priority. This system makes the algorithm solve one such problem at a time: all new conflicts that arise after solving the obstruction conflict have a higher priority than the other obstruction(s). In doing so, the depth of the search tree is limited: if the first obstruction conflict cannot be solved, it is of no use to look at the other ones.

No tiebreaker is added to the sorting process, which means conflicts retain their original respective order in case their priorities are equal. The original order depends on the order of the train patterns to which they relate. One of the business requirements for the VGB solver is to treat trains from different operators equally. This is implemented by sorting trains by their train number, as those numbers are not based on the operating company. Some experiments were performed with changing this order to separate trains based on their type. Prioritising conflicts for intercity trains over those for local trains or vice versa showed some minor improvement, but the experiments described in chapter 4 resulted in a much better performance increase. Furthermore, after the improvements described in that chapter were implemented, changing the conflict order no longer had any effect on the performance.

3.3. Evaluating results

A solution to the problem is reached once a node is found that has no more conflicts. Applying all adjustments used to reach that node in the tree leads to a stable alternative schedule. However, it is important to check if that solution is actually good. Simply cancelling all affected trains is always a solution, but rarely the desired outcome.

To determine the quality of a set of adjustments, three perspectives are taken into account:

- Inconvenience for passengers,
- Workload for train traffic controllers,
- Robustness of the alternative schedule.

Adjustments and results thereof are scored according to the impact they have on these topics. The higher the impact, the higher the number of penalty points. Different solutions can be compared with each other using this scoring system. However, one scenario can be more difficult than another, causing the best solutions for each to still have wildly different scores. As such, only scores of solutions for the same scenario should be compared.

3.3.1. Focus set

Different scenarios can have very different characteristics. For example, some scenario could be a partial blockage on a track with only three trains per hour, while another could entail a full blockage of all tracks leading to a very busy station. Therefore, changes to the algorithm can improve the score for some scenarios, while having a detrimental effect on others. To evaluate the result of a change properly, its impact on different scenarios should be taken into account. However, it is not manageable nor desirable to run the solver on all available cases. Therefore, a set of around 30 scenarios is used for most testing purposes. The scenarios in this so-called “focus set” are representative in terms of geographic location, types of blockages and number of affected trains.

The solver is developed in stages, and each stage has an associated milestone. Each milestone describes which types of conflicts should be supported. As such, scenarios get increasingly more complex with every milestone. The focus set contains scenarios for each milestone that has been worked on or passed earlier, to provide a representative set to test with.

For more complicated scenarios, the solver may need more than 24 hours to process all nodes in the search tree. Since this would lead to a slow feedback loop, the run time for tests is typically limited to 30 minutes. To evaluate whether this time limit has a significant impact on the quality of the solutions, a run with a time limit of 24 hours is performed once a week.

3.3.2. Interactive solver

The CGI team developed an interactive version of the solver, where the user can manually enter the adjustment to take at each node. Using the interactive version, paths can be taken that are not considered by the automatic solver yet, due to time constraints or viability of the path according to the heuristic value. With that process, the adjustments given by the scenario makers can be replayed to find the score of that solution. Using that score as a baseline gives a good indication of whether the solution found by the solver is good enough.

3.4. Optimisations

Using a search tree to evaluate all options may still lead to unrealistic computation times, as the number of different options can be immensely large. This section describes the most important optimisations that are already implemented in the solver, so even somewhat more complex scenarios can be dealt with.

3.4.1. Pruning

As is typical for a branch and bound algorithm, some branches of the search tree are pruned. Once the first conflict-free node is found, its score can be used as a threshold. Any other node that already scores worse than this threshold can be eliminated: to solve its remaining conflicts, more adjustments have to be made and this will make the score even worse.

The main problem with this optimisation is that it does not work well for incomplete subtrees. A node that still has conflicts is likely to have fewer adjustments than a node without conflicts, which means that the score of the former node is lower. Nodes are only cut when their actual score is above the threshold. Ideally, it would be possible to estimate how many points it would cost to solve the remaining conflicts. That way, branches could be eliminated earlier in the process. However, finding a formula for a good estimate of the score of further adjustments has proven to be a great challenge for the team. The difficult part is that the same type of conflict may have to be solved in different ways in different cases, depending on the affected infrastructure and other trains. As such, no estimate is used currently. Given that the team already tried using estimations and was unsuccessful, this optimisation is not looked into in the remainder of this thesis.

3.4.2. Heuristic value

Most of the time, multiple nodes could be chosen as the next one to process. Various strategies exist to make this choice. Well-known examples are breadth first (first in, first out), depth first (last in, first out), and lowest cost first [22]. With the first two options, trees will always be created in the same order for different problems. However, since “branch and bound algorithms can be (and often are) slow” [23] and the number of potential solutions is immense, not all nodes can be computed in a reasonable time frame. With breadth first order, the tree would be processed layer by layer and therefore solutions deep in the tree may never be found. Conversely, with depth first order one branch of the tree will be processed until it no longer has conflicts (or becomes unsolvable), only then will the next branch be taken up. This order gives a very low exploration rate, potentially missing paths to good solutions early on.

The third option is based on some cost function. This function should determine the probability that that node leads to a good solution. With that method, the way the tree is traversed is dynamic: it can vary from one scenario to the other.

Using the lowest cost first order only works well if the cost function is defined properly. At the time of writing, the heuristic value is based on the number of remaining conflicts, the number of remaining obstruction conflicts (as these are the most difficult ones, they are weighed separately), the number of train patterns that still have at least one conflict, the number of patterns that still have to be mapped after short-turning somewhere, and the score of all adjustments required so far. That formula has been the same for years, and the current development team is not fully aware of the reasoning behind it. No actions taken to improve the formula have been documented, and the formula is not actively worked on by the development team. There could be room for improvement and experiments would not interfere with active development, which is why this part is analysed and experimented on later in this thesis.

3.4.3. Similar nodes

When processing a node, child nodes are created for all different ways to solve the most important conflict. Many nodes with similar adjustments can be created in this way, for example two nodes that both represent rerouting a train, but using a different route. As the number of nodes could grow enormously, these new child nodes are grouped if they perform a similar adjustment. For example, it is checked if both changed the routes of the same trains (regardless of the exact route), or if they cancel the same train series. Nodes are only grouped if they also have similar remaining conflicts: one should be a subset of the other. From a group of nodes, only the node with the lowest score will be kept, while the rest is discarded. Other attempts to prevent the creation of similar or duplicate nodes are actively developed by the team. For that reason, this thesis does not look further into this method of optimisation.

4 Approach and Results

This chapter describes various experiments and their results, with the aim of improving the solution quality of the VGB solver. For relatively simple scenarios, the solver can process all nodes well within the 30-minute run-time limit of the tests. This means the best solution will be found consistently. Unfortunately, this is only the case for a third of the focus set. For the other tests, the solver runs out of time to process all nodes. Therefore, the best solution may never be found: its node can be one of the remaining nodes. It can even be the case that none of the solutions are found, if all corresponding nodes are still to be processed after 30 minutes.

There are two ways to solve this problem: increasing the number of nodes that are processed, or improving the order in which the nodes are processed. The first option requires detailed knowledge of the inner structure of the existing code base. Development team is continuously working on this process. Since diving deeply into the internals of the solver would cost a lot of time, performance optimisation in this manner was deemed out of scope for this thesis. The second option, improving the order in which the nodes are processed, can be implemented by changing the calculation for the heuristic value (see subsection 3.4.2).

This chapter explains various experiments regarding the calculation of the heuristic value. First, the methods for comparing results are explained in section 4.1. Then, section 4.2 describes experiments in which the calculation of the heuristic value was optimised. Lastly, the application of a machine learning classification strategy is explained in section 4.3.

4.1. Comparing results

As explained in section 3.3, a scoring system that takes multiple perspectives into account is implemented in the solver. Using this, different solutions for the same scenario can be compared. However, finding a solution with a lower score does not guarantee that the adjustments are good enough to be used in practice. As explained in subsection 3.3.2, a baseline score can be found using the interactive solver. This gives a better intuition of whether a solution could be used in practice, but no guarantee. Furthermore, alternative solutions with a similar overall impact can have varying scores.

Scenario makers are sometimes consulted to check if another solution found by the solver could be viable in practice. This feedback loop can take quite some time and involves manual work, which is not ideal for checking many different solutions for various scenarios. Even though it would be ideal to compare how many solutions are good enough for practice, given the difficulties for that process, results for the experiments in this thesis are compared solely based on their scores.

4.2. Changing heuristic value calculation

As explained in subsection 3.4.2, the solver uses a heuristic value to determine which node to process next. Changing the calculation for this value changes the order in which the nodes are processed.

The original formula is as follows:

$$(1 + P + L) \times (1 + O + L) \times S + C \tag{4.1}$$

Using the following definitions:

- P: The number of unique patterns with at least one remaining conflict.
- L: The number of unique patterns that are adjusted using short-turning, but are not yet reconnected to a pattern for the way back (this loop should be closed).
- O: The number of obstruction conflicts (where a train passes through an obstructed area) that remain.
- S: The penalty points for all adjustments made to get to the current node.
- C: The number of conflicts that still need to be resolved.

For all these terms, lower values are better.

The calculation was left unchanged for quite some time, but none of the team members knew why this exact formula would work well. Despite the unknown origin, some components of the formula intuitively seemed useful to determine how promising a node is. For example, the more conflicts a node has remaining, the more complex it will be to solve, assuming that the conflicts are of equivalent difficulty. This intuition turned out to be correct in an initial, accidental experiment.

4.2.1. Single factors

The first change made to this formula is to use only the number of conflicts. The intention was to create a simple but impactful change, to try the general impact of changes to the formula. It was expected that this new formula would perform significantly worse. At that time, no solution could be found for 15 scenarios of the focus set, which then consisted of 36 scenarios. With the new formula, two scenarios got a significantly worse score. However, for 14 of the aforementioned 15 scenarios without any solution previously, a solution was found. The development team immediately noticed this massive improvement and looked into it. As mentioned earlier, it is to be expected that a node with more conflicts is more difficult to solve, in case of equivalent conflicts. To incorporate the difficulty of the conflicts in the formula, the score was added as tiebreaker.

More experiments were performed using the other terms of the original formula. Using only the score was hypothesised to give good results, but this turned out to be incorrect. Initially, this seemed odd, as it correlates with the number of conflicts, and it seemed reasonable to assume that adding the weights of the conflicts to the equation would be even better. However, some nodes with many low-weight conflicts got worse scores than nodes with only a few high-weight conflicts, even though the former option could be solved much faster. This could mean that the scoring system is not completely accurate, and that revising it could lead to better results. On the flipside, there is no method of comparison after changing the scores: simply reducing the number of points for all adjustments obviously leads to lower scores, but the solution itself may not have changed or even become worse. Another solution could be to separate the scores of adjustments in the heuristics calculation from those used for the final score. In addition to the significant code changes required to implement this, it would take a lot of work to tune the parameters.

Using any of the other terms of Equation 4.1 directly as the heuristic value had only a negative impact. The results of the formula with the number of conflicts, with a score as a tiebreaker, were much better than before and showed only very limited negative effects.

Tweaking the formula

At the time of performing these experiments, the solver sometimes progressed very deep into the search tree, even though this was not necessary at the relevant milestone. To prevent the solver from going too deep, some penalty points for going over a depth of 50 were added to the formula of the previous section. Furthermore, if some of the hardest types of conflicts are still present at that depth, the penalty is doubled.

The final formula was as follows:

$$\text{Heuristic value} = \begin{cases} C & \text{if } D \leq 50 \\ C+3 & \text{if } HC = 0 \\ C+6 & \text{otherwise} \end{cases} \tag{4.2}$$

Using these definitions:

- C: The number of conflicts that still need to be resolved.
- D: The depth of the node.
- HC: The number of hard conflicts.

Ties were broken by the score of the node.

Results

The results of the new formula for the heuristic value had a big impact on the performance of the solver. Generally, solutions were found for many more scenarios than with the old formula. For scenarios that already had a solution, newly found solutions were of much better quality.

To show the general impact, a periodic performance analysis is used. This includes other changes made by the development team, but the majority of the performance change was caused by the new heuristic value formula. The periodic performance analysis performed by the CGI team is shown in Table 4.1. In that analysis, the judgement of "good enough" is given based on a deviation from the estimated score.

		Nov	Jan
Milestone 1	Total	66	66
	Found solutions	66	66
	Good enough	54	54
Milestone 2	Total	78	78
	Found solutions	70	77
	Good enough	37	40
Milestone 3	Total	249	249
	Found solutions	133	233
	Good enough	52	58
All milestones	Total	962	962
	Found solutions	425	832
	Good enough	213	235

Table 4.1: Results when using Equation 4.1 or Equation 4.2 for the heuristic value. Differences are highlighted.

As milestone 1 contains mostly simple scenarios, many solutions are found well within the time limit of 30 minutes. Since the solver can process all nodes of those scenarios, changing the processing order has no effect on them. Milestones with more complex scenarios show more impact when using the new formula.

On the normal 30-minute testing process, the penalty point system improved the solutions for two of the scenarios in the focus set. For longer runs, it improved the results of other scenarios as well.

4.2.2. Depth

As described earlier, the solver sometimes progressed too deep into the search tree. Several experiments were conducted using the depth in the heuristic value formula in some way, to penalise this behaviour. The starting point was the formula from the previous section, with the penalty system for depths greater than 50 removed. To prevent the solver from going too deep into the tree, the depth of the node was added as a term directly. This gave large improvements for several scenarios, but had a detrimental effect on others.

As a second experiment, the number of hard conflicts was added to the formula, as it was also in the penalty system before. Again, the results improved for some scenarios and worsened for others, even compared to the scores of the previous experiment. In general, these scores were better than using only depth, but there was no significant improvement over Equation 4.2. Using the definitions of terms provided earlier, the resulting formula was as follows:

$$C + D + HC \tag{4.3}$$

Ties were still broken by the score of the node.

Expected depth

It was noticeable that for scenarios in which it had a negative effect, the solver found the solutions deeper in the tree. Trying to limit the depth meant that the solver would explore more, but did not reach the nodes with the solution(s) found previously. A new idea came to mind: if it is possible to determine the expected depth to find solutions, penalising the depth could be done relative to that.

The following data was obtained for all scenarios:

- Number of blocked patterns,
- Number of affected patterns (patterns that are use either blocked tracks, or tracks parallel to blocked tracks),
- Number of routes blocked (every route is a different way to get from one end to the other end of the affected area),
- Number of routes affected (similar definition of affected as above),
- Number of indirectly affected patterns (patterns that are not traversing the blocked area, but stop on the station at either end of the incident area),
- Number of train loops of which at least one pattern is blocked.

Furthermore, the internal search trees for all scenarios in the focus set were analysed to retrieve the depth of all solutions found by the solver.

The correlation coefficient between the number of blocked patterns and the average depth of all solutions per scenario was 0.72, the highest of all data points. The average depth at which solutions were found was 4.19 times the number of blocked patterns. The standard deviation of this factor is 3.42, which is relatively large. However, using only the depth of the best solution rather than the average of all solutions per scenario, the standard deviation was reduced to 1.62, while the ratio remained the same. The correlation then improved to 0.80.

This expected depth was used for further experiments. These experiments included:

1. Replace depth in the formula by the depth of the node relative to the expected depth (minimum 0)
2. Replace depth in the formula by the depth of the node relative to half of the expected depth (minimum 0)
3. Replace depth in the formula by the depth of the node relative to ¾ of the expected depth (minimum 0)
4. Same as 1, but also add the number of critical conflicts starting from the expected depth
5. Always have depth in the formula, and add it another time starting from the expected depth

However, none of these experiments were clearly better than only adding depth as a term. Here again it stood out that the effects of these changes were very good for some scenarios, while others received much worse scores than before. For some scenarios the depth made too big of an impact, so more experiments were conducted. In those, the depth was put in the formula multiplied by a factor ranging from 0 to 1 in steps of 0.1. Therefore, the formula used was the following.

$$C + HC + RD \times F \tag{4.4}$$

In this equation, RD is the relative depth, and F is the weight factor. Ties are broken by the score of the node. Figure 4.1 displays the relative score changes for each scenario in the focus set.

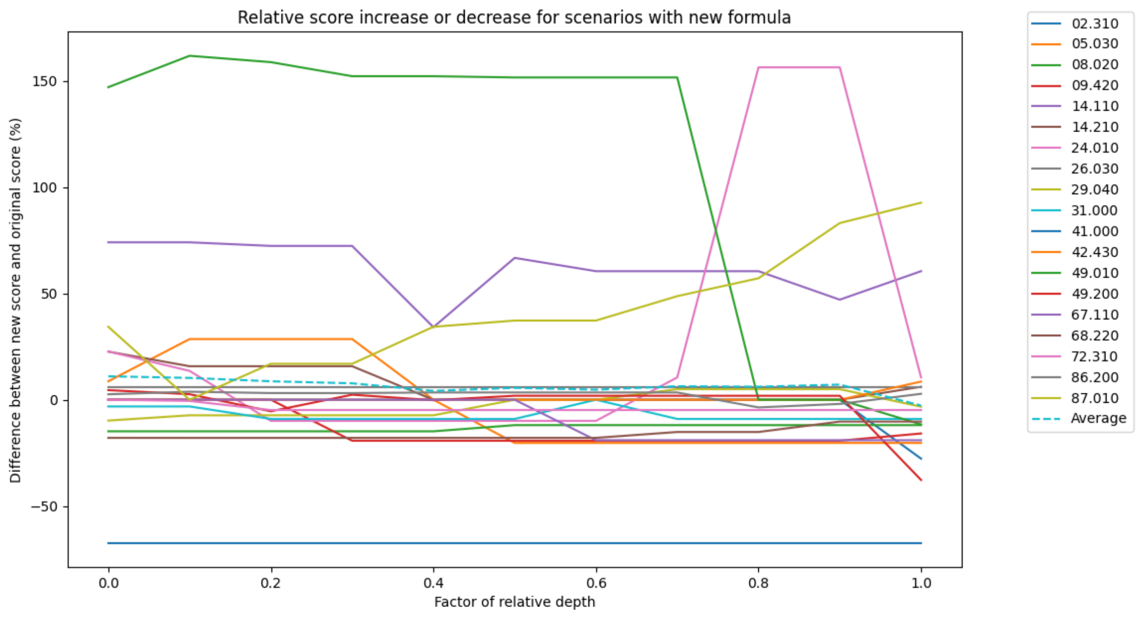

Figure 4.1: Relative score increase or decrease by using Equation 4.4 compared to Equation 4.2.

The scores for several scenarios became worse altogether compared to using the old formula. Figure 4.2 gives a more detailed look at the results for the majority of the scenarios, by removing the six scenarios whose scores are most different.

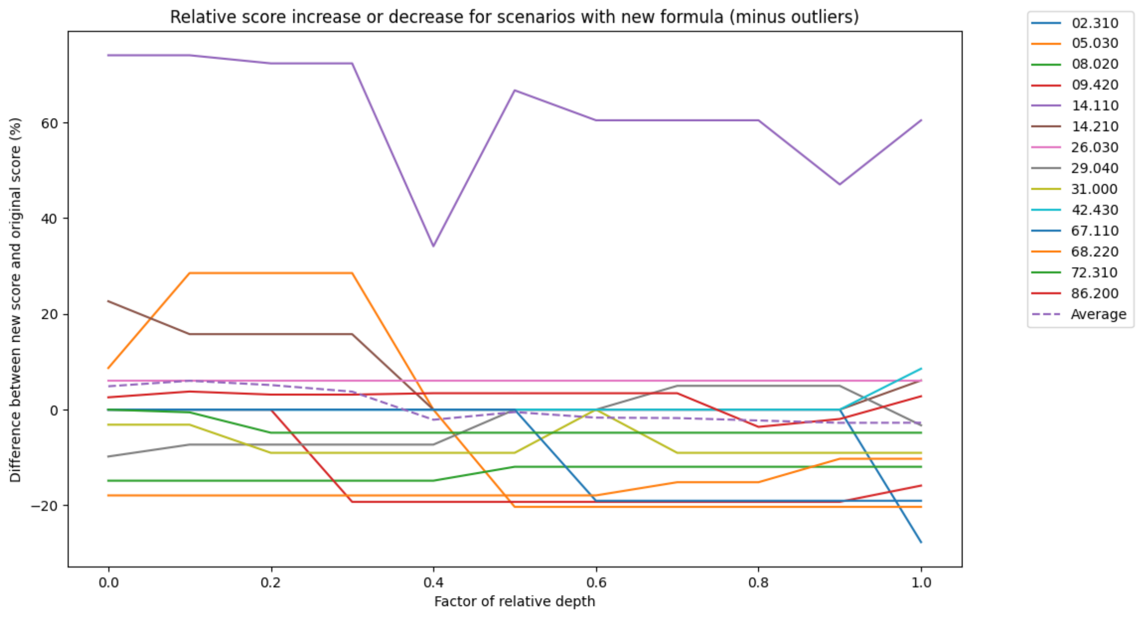

Figure 4.2: Figure 4.1 minus the six outliers for better overview. The average score is lowest using a weight of 0.4.

It is now clear that for the remaining scenarios, the best results are found around the factor of 0.4.

Even though the average score was much worse when the outliers were included, the minimal point was at the same weight value. Only one of the removed outliers had a better score, all others had significantly worse scores for most weight values. A significant number of scenarios still suffers from worsened scores at the optimal weight factor. As the results were not better than those of Equation 4.2, that formula remained in the solver.

4.3. Classifying scenarios

All experiments showed potential for several scenarios, but also had adverse effects on others. If one can determine the effect of a new formula on the scenario up front, a different strategy could be applied case by case, filtering out the negative impact as much as possible.

To check whether selecting from different strategies would be viable, the formula of Equation 4.3 was used, as it showed the greatest potential for positive impact. The data points from Equation 4.2.2 were used to build a classification system. To improve the quality of the classifier, this data was extracted for all 1,000 available scenarios, rather than only those in the focus set. Furthermore, ratios between several of these data points were computed and added to the data file.

Scenarios that receive better results with the new formula are assigned 1 as label, the ones that became worse are assigned -1. Scenarios on which the new formula had no effect, those with label 0, were dropped from the data set. Misclassifying these would have no effect on the overall scores after all. The remaining data was split 50/50 into training and test sets. Then, SKLearn's DecisionTreeClassifier was run, with a maximum allowed depth of 3 and at least 5 samples per leaf node. The depth parameter was chosen by running several experiments with varying depths, ranging 1 to 10. Figure 4.3 shows the accuracy at these values. After a depth of 3, the accuracy on the training set increased, but the classifier performed worse for the test set. That difference in accuracy is a clear sign of overfitting. Limiting the depth prevents it from occurring. Another benefit is that the resulting decision tree is very explainable and can even be coded into the existing solver, thereby not requiring an external classification system. The accuracy of the classifier is 73.5% on the test set.

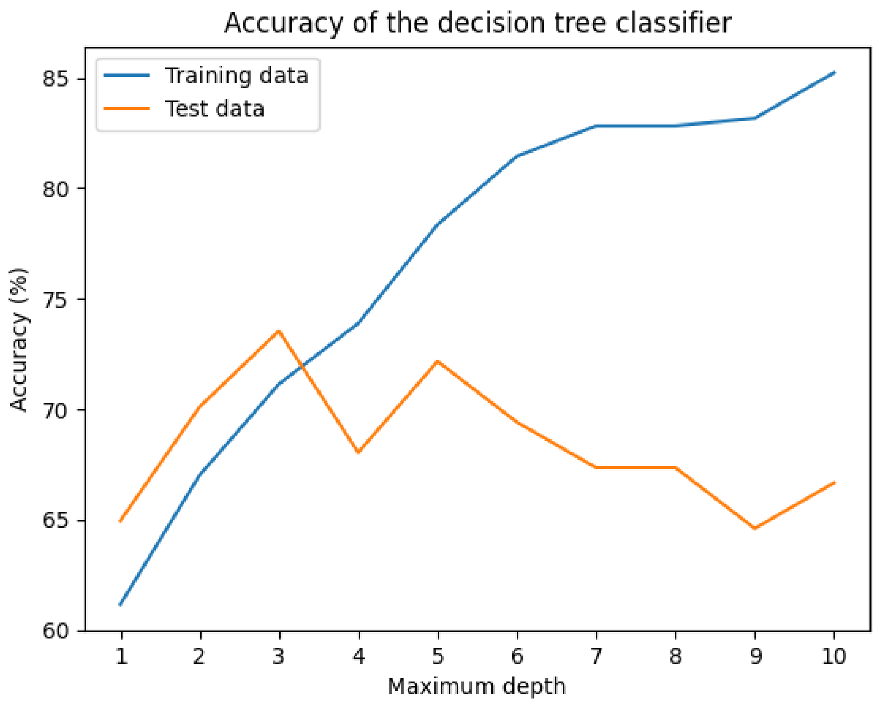

Figure 4.3: Accuracy of the decision tree classifier using different maximum depth values. Accuracy on the test data is highest at a maximum depth of 3.

Additionally, a random forest classifier was created using the same data. Even after optimising its parameters, the difference in accuracy is only 1 to 2 percent points. Given the advantages of the decision tree classifier described above, that would be the preferred approach.

4.3.1. Impact on scores

To evaluate the impact of using the decision tree, the resulting scores of all scenarios in the megarun were calculated as if the classifier were used. The new score was taken if the label was predicted to be 1, otherwise the old score was used. Although 271 scenarios received better scores without the new formula, only 80 of those remained after classification. On average, these scored 16.7% worse than before. Of the 301 scenarios with improved scores with the new formula, 153 were correctly classified. Their scores improved by 17.5% on average. For 11 scenarios, the solver found a solution using the new formula, whereas using the old formula did not result in one. Six of them remained after classification. However, of the 82 scenarios that no longer resulted in a solution using the new formula, only three are classified incorrectly. This shows there is great potential for improvement by using a new formula for the heuristic value, if an accurate classification system is used to select the scenarios on which to apply said formula.

5 Conclusion and Future Work

In this chapter, the research question from chapter 1 is answered. After that, recommendations and future work are discussed.

5.1. Conclusion

In chapter 1, the research question was introduced: how can the solution quality of the VGB Solver be improved? Given the structure of the VGB Solver, improvements could be made in two ways: processing more alternative options, or processing the same options in a better order. The former option requires detailed insights into the internal workings, which was not feasible for this thesis. The latter option was more accessible and provided ample opportunity for experiments. The scope was limited to small but potentially impactful changes, so the experiments could fit the duration of this thesis while remaining compatible with ongoing development work. Using these experiments, the sub-questions of the research question can be answered.

How can the performance of the VGB Solver be improved by leveraging the existing optimisations better?
The majority of the experiments conducted during this project related to the calculation of the heuristic value of nodes. As that value is used to determine the processing order of the nodes in the internal tree, changing it can dramatically alter the performance. One of the attempted new formulas, using the depth of a node as the primary value, improved the performance significantly: solutions were found for 96% more scenarios than before. This formula was integrated into the solver by the development team after some minor tweaks. Further experiments on the heuristic value computation showed potential, but often faced the drawback of significantly worsening scores on several scenarios.

How can data about specific problem instances be used in the VGB Solver?
Scenario makers sometimes treat scenarios differently, as they can tell from experience that certain adjustments will not work well (enough). For example, they will not try to fit all trains into the schedule on a very busy line in case half of the tracks are unavailable. The VGB solver used the same strategy for all scenarios, which caused tweaks to the algorithm to work well for some scenarios, but have a negative impact on others at the same time. To allow different strategies for different scenarios, a classification algorithm was implemented. The classifications were made based on data of the surrounding infrastructure and timetables of directly and indirectly affected trains. This resulted in a decision tree of three levels, that categorised the scenarios into positive and negative impact for a specific heuristic value formula with 73.5%, disregarding scenarios on which said formula had no impact. By choosing which of the two available formulas to apply based in the classifier, the number of scenarios that improved was roughly twice as high as those that deteriorated. The relative improvement per scenario was higher than the loss, making the results mostly positive.

In summary, the solution quality of the VGB Solver was greatly impacted by replacing the formula to calculate the heuristic value of a node. Furthermore, experiments using a decision algorithm to apply a different formula for different types of scenarios showed promising results.

5.2. Recommendations/future work

Despite the numerous formulas that have been tried to compute the heuristic value, many more combinations of the properties of nodes are possible. The depth of nodes was found as an important factor, but on some scenarios this had adverse effects. The experiments using the “expected depth” were promising, but that formula did not perform better than when using the regular depth. From these experiments, it was concluded that depth penalties above the expected depth have a positive impact on the processing order. Further experiments could be conducted to combine the actual depth with the depth relative to the expected depth of the node to see if these two together can lead to even better results.

The formula and decision tree found in section 4.3 could be tested with the latest version of the VGB solver. It would be even more interesting to see if other strategies can be found that work well for a different set of scenarios. If a classifier also works well for those strategies, the solver could pick from multiple different strategies for each scenario.

Taking other factors into account for the heuristic value could also be an option. For example, the solver could check if there was any significant progress in the last couple of nodes leading up to the current one. If not, perhaps it would be better to process other nodes first.

Another interesting direction to look into is reducing strictness. This would mean the solution produced by the solver would not have to be conflict-free. Such experiments were not conducted for this thesis, as it is hard to predict how useful a solution with conflicts is: solving even a single remaining conflict can create many more conflicts. However, another approach could be used. Many conflicts involve time limits, for example that two trains cannot cross the same switch in opposite directions within a specified amount of time. The strictness of these constraints could be reduced, which could allow trains to be closer than the indicated times. While this may solve some conflicts and is also used in practice, the exceedance of the time constraint should be minimised. Further research could indicate to what extent reducing this strictness could be useful in obtaining solutions faster.

For the solver itself, it would be good to look at the scoring system. Currently, the quality of solutions are determined based on points for certain adjustments and consequences thereof. This score is also used in the new heuristic value formula. This makes it hard to adjust the weights for certain actions for the solver: if the scores are adjusted, the results are no longer comparable to previous results. Separation of these scores ensures that they can be adjusted for use in the heuristic formula, providing more room for experiments and potential improvements. Furthermore, the scores used to evaluate the solution quality should be double-checked, so lower-scoring solutions are actually most desirable, as this is not always the case currently.

Lastly, a general recommendation for other projects that aim to automate manual tasks. Designing an algorithm to solve all instances of the problem in the same way may not be the best solution. Using the experience of the people performing the manual task, multiple strategies can be developed, and representative data points can be found. Then, by applying machine learning algorithms, it can be determined which strategy should be applied when.

References

[1] Autoriteit Consument & Markt. *ACM Rail Monitor: The Netherlands has Europe's busiest railway network*. Mar. 2019. URL: https://www.acm.nl/en/publications/acm-rail-monitor-netherlands-has-europes-busiest-railway-network.

[2] ProRail B.V. *Op zoek naar Een Unieke uitdaging met impact? Solliciteer dan bij ProRail verkeerleiding en laat de Nederlandse treinen veilig en op tijd rijden.* URL: https://www.werkenbijprorail.nl/vakgebieden/verkeersleiding.

[3] Jing-Quan Li, Pitu B. Mirchandani, and Denis Borenstein. "The vehicle rescheduling problem: Model and algorithms". In: *Networks* 50.3 (2007), pp. 211–229. DOI: 10.1002/net.20199. URL: https://doi.org/10.1002/net.20199.

[4] Rodrigo Acuna-Agost et al. "A MIP-based local search method for the railway rescheduling problem". In: *Networks* 57.1 (Dec. 2010), pp. 69–86. DOI: 10.1002/net.20384. URL: https://doi.org/10.1002/net.20384.

[5] G. Caimi et al. "A New Resource-Constrained Multicommodity Flow Model for Conflict-Free Train Routing and Scheduling". In: *Transportation Science* 45.2 (May 2011), pp. 212–227. DOI: 10.1287/trsc.1100.0349. URL: https://doi.org/10.1287/trsc.1100.0349.

[6] Wei Liu, Xiaoning Zhu, and Liujiang Kang. "Real-Time Track Reallocation for Emergency Incidents at Large Railway Stations". In: *Mathematical Problems in Engineering* 2015 (2015), pp. 1–11. DOI: 10.1155/2015/296394. URL: https://doi.org/10.1155/2015/296394.

[7] Sander Van Aken, Nikola Bešinović, and Rob M.P. Goverde. "Designing alternative railway timetables under infrastructure maintenance possessions". In: *Transportation Research Part B: Methodological* 98 (Apr. 2017), pp. 224–238. DOI: 10.1016/j.trb.2016.12.019. URL: https://doi.org/10.1016/j.trb.2016.12.019.

[8] Sander Van Aken, Nikola Bešinović, and Rob M.P. Goverde. "Solving large-scale train timetable adjustment problems under infrastructure maintenance possessions". In: *Journal of Rail Transport Planning &amp Management* 7.3 (Dec. 2017), pp. 141–156. DOI: 10.1016/j.jrtpm.2017.06.003. URL: https://doi.org/10.1016/j.jrtpm.2017.06.003.

[9] Ilse Louwerse and Dennis Huisman. "Adjusting a railway timetable in case of partial or complete blockades". In: *European Journal of Operational Research* 235.3 (June 2014), pp. 583–593. DOI: 10.1016/j.ejor.2013.12.020. URL: https://doi.org/10.1016/j.ejor.2013.12.020.

[10] Nadjla Ghaemi, Oded Cats, and Rob M.P. Goverde. "A microscopic model for optimal train short-turnings during complete blockages". In: *Transportation Research Part B: Methodological* 105 (Nov. 2017), pp. 423–437. DOI: 10.1016/j.trb.2017.10.002. URL: https://doi.org/10.1016/j.trb.2017.10.002.

[11] Patrick Looij. "Adjusting train routing in case of planned infrastructure maintenance". In: (Sept. 2017). URL: https://repository.tudelft.nl/islandora/object/uuid%3A96ae098d-5708-45c4-9270-c76f728d104e.

[12] Christian Liebchen and Hanno Schülldorf. "A collection of aspects why optimization projects for railway companies could risk not to succeed – A multi-perspective approach". In: *Journal of Rail Transport Planning &amp Management* 11 (Oct. 2019), p. 100149. DOI: 10.1016/j.jrtpm.2019.100149. URL: https://doi.org/10.1016/j.jrtpm.2019.100149.

[13] Ryo Kuroiwa and J. Christopher Beck. *Domain-Independent Dynamic Programming: Generic State Space Search for Combinatorial Optimization*. 2023. arXiv: 2211.14409 [cs.AI].

[14] Sean R Eddy. "What is dynamic programming?" In: *Nature Biotechnology* 22.7 (July 2004), pp. 909–910. DOI: 10.1038/nbt0704-909. URL: https://doi.org/10.1038/nbt0704-909.

[15] Irving Van Heuven Van Staereling. “Tree Decomposition Methods for the Periodic Event Scheduling Problem”. en. In: (2018). DOI: 10.4230/OASICS.ATMOS.2018.6. URL: http://drops.dagstuhl.de/opus/volltexte/2018/9711/.

[16] L.G. Kroon, D. Huisman, and G. Maróti. *Railway timetabling from an operations research*. Econometric Institute Research Papers EI 2007-22. Erasmus University Rotterdam, Erasmus School of Economics (ESE), Econometric Institute, June 2007. URL: https://ideas.repec.org/p/ems/eureir/10346.html.

[17] Alison Eele and Arthur Richards. “Path-Planning with Avoidance Using Nonlinear Branch-and-Bound Optimization”. In: *Journal of Guidance, Control, and Dynamics* 32.2 (Mar. 2009), pp. 384–394. DOI: 10.2514/1.40034. URL: https://doi.org/10.2514/1.40034.

[18] David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge, England: Cambridge University Press, Apr. 2011.

[19] Theo C. Ruys. “Optimal Scheduling Using Branch and Bound with SPIN 4.0”. In: *Model Checking Software*. Ed. by Thomas Ball and Sriram K. Rajamani. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 1–17. ISBN: 978-3-540-44829-7.

[20] Alison Eele and Arthur Richards. “Multi Vehicle Avoidance Using Nonlinear Branch and Bound Optimisation”. In: *AIAA Guidance, Navigation, and Control Conference*. American Institute of Aeronautics and Astronautics, June 2009. DOI: 10.2514/6.2009-5780. URL: https://doi.org/10.2514/6.2009-5780.

[21] Mohammad Tamannaei and Iman Irandoost. “Carpooling problem: A new mathematical model, branch-and-bound, and heuristic beam search algorithm”. In: *Journal of Intelligent Transportation Systems* 23.3 (Nov. 2018), pp. 203–215. DOI: 10.1080/15472450.2018.1484739. URL: https://doi.org/10.1080/15472450.2018.1484739.

[22] Anurag Dutta et al. “A Review on Optimality Investigation Strategies for the Balanced Assignment Problem”. In: *2023 International Conference on Computational Intelligence and Sustainable Engineering Solutions (CISES)*. 2023, pp. 254–259. DOI: 10.1109/CISES58720.2023.10183493.

[23] Stephen Boyd and Jacob Mattingley. “Branch and bound methods”. In: *Notes for EE364b, Stanford University* 2006 (2007), p. 07.