Erasmus University Rotterdam [1]

Erasmus School of Economics

Master Thesis

# A greedy randomized adaptive search procedure for freight train timetabling in two-way networks with mixed single and double-track segments and passing loops.

*Author*

Rob Werst (412215)

*Supervisor*

Dr. Shadi Sharif Azadeh

*Second assessor*

Dr. Twan Dollevoet

*Date final version:*

December 17, 2020

[1]The views stated in this thesis are those of the author and not necessarily those of Erasmus School of Economics or Erasmus University Rotterdam.

**Abstract**

Train timetabling is an important step in efficient railway transportation planning. In this thesis a new exact model is established and variable reduction and bound restriction techniques are applied to reduce the computation time needed to solve it. These techniques can lead to a decrease in computation time of up to 95% in some instances. The improved exact formulation can solve instances up to 100 trains to (near) optimality within hours of computation time. A GRASP is designed to solve large real-life instances. Within one hour of computation time feasible timetables with a gap of at most 6% on average can be constructed using the GRASP for instances up to 100 trains. For a large real-life instance consisting of 227 trains a gap of at most 26% and an expected gap of around 10% can be achieved given 10 hours computation time. A method for bottleneck identification is added to the GRASP to provide an indication of which segments could benefit most from an extra piece of track. Adaptations are discussed to make the GRASP applicable in case of delays and disruptions. One of the main advantages of the GRASP is that it allows for complete parallel implementation. Thus given enough computing power, a large number of iterations could be performed in limited time. It is concluded that the GRASP constructs adequate timetables for both small and large real-life problems and can be regarded as an interesting addition to current heuristics for the train timetabling problem.

# Acknowledgements

I would like to thank Ab Ovo for giving me the opportunity to do my master thesis research with them during these turbulent times. In particular I would like to thank Wouter Merkx for his constructive feedback, guidance and help during my internship. In addition I want to thank Dr. Shadi Sharif Azadeh for supervising this thesis and for our pleasant feedback sessions. Finally, I cannot in good conscience finish these acknowledgements without thanking Caro and Martin Werst, Taha Boujnane, Ruben van den Broek, Stijn Lucieer and Emma de Koning for the countless cups of coffee we drank together and heaps of moral support over the past 6 months that were of paramount importance for my success, productivity, and sanity.

# Contents

# 1 Introduction

As global production levels among virtually all sectors continue to rise, efficient access to production goods and ways to easily distribute finished products are more important than ever. A considerable amount of cargo worldwide is moved by rail. Depending on the region this can be between 5 and 50% of total cargo haulage, according to Eurostat. Efficient use of railway networks is crucial to keep transportation cost low, yet in large railroad networks this proves to be a considerable challenge. The logistical problem of railway transportation is typically divided up in the following stages: First, the demand needs to be determined and sufficient rolling stock must be made available to accommodate it. Next, loads are assigned to specific trains in a way that efficiently utilizes the trains capacity and the nature of the load. Then, a timetable must be constructed that assures the railway network is used in such a way that unnecessary delays are kept to a minimum and no collisions occur in the network. The objective of this step may differ depending on whether passengers or goods are being transported. After the construction of a timetable, crew and shift assignments are made to conclude the transportation planning process.

In this research the focus will be on train timetabling. It is of paramount importance that this step in the transportation planning process is executed precisely, as a single wrong track assignment or departure time could have disastrous consequences for an entire schedule. There are a few key distinctions one can make to specify train timetabling problems. Is the timetable to be made cyclical? And if so, what is the period? Or is the set of trains to be scheduled changing continuously? A cyclical schedule is more often encountered in passenger transport, but can also occur in cargo transportation, although is it more common to have larger periods. The next big distinction is passenger vs cargo transport. Cargo transport planning is often less affected by small delays than passenger transport. Usually passenger transport will have more frequent stops as well as shorter stopping times than cargo transport. Passenger transport will usually take priority over cargo transportation if there is heavy traffic for both. In many European networks this means that cargo must be moved mostly outside of peak hours for passenger travel. In more spaced out networks, on the North American continent for example, this is less of an issue as on large sections of the network there is barely any passenger traffic. Finally, there is the topic of network topology. The complexity of the problem changes drastically depending on whether the entire network consists of single-track segments, whether or not there are passing loops, double-track segments, or multiple main tracks in general. The main goals of this research is to construct and benchmark a greedy randomized adaptive search procedure(GRASP) to solve a cargo variation of the train timetabling problem. The constructed schedule will be cyclical, meaning that it can be repeated every period. The network under consideration consists of both single and double track sections and includes passing loops at which trains can wait to be overtaken.

Besides effective timetabling, network extensions could be considered to increase efficiency and prepare for an expected increase in demand. The network on a corridor can be extended by adding an extra track to a section or adding some passing loops. The main question is where in the network a piece of extra track would be most effective. A good candidate could be a part of the network that given a timetable has a lot of trains passing each other. Extending the network on such a "bottleneck" could allow for a new timetable in which trains will have to wait less. In this thesis a method will be proposed to assist in selecting portions of the network that could benefit from an extra piece of track. As delays and maintenance occur often in real life cases, some information will also be provided on how to adjust the Algorithms proposed in this thesis to deal with these disturbances.

This thesis is part of an internship at Ab Ovo. Ab Ovo is a consultancy company based in the Netherlands that is specialized in logistic and supply chain optimization solutions. The problem tackled in this thesis is inspired by a case provided by Ab Ovo based on a railway network on the North American continent.

In section 2 literature relevant to the problem and the proposed solution methods is discussed as well as the relevance of the solution techniques applied in this thesis. In section 3 a clear problem description is presented, followed by the research questions based on this problem. In section 4 the methodology is presented. The GRASP algorithm and its parts are discussed in detail, an exact model for reference is established and some ideas for variable reduction are discussed as well as the aforementioned bottleneck identification method. In section 5 relevant results are presented after applying both the exact model and the GRASP to test instances and run-time and solution quality are compared. Relevant meta-parameter choices will also be discussed in this section. Afterwards the results of a real-life case study provided by Ab Ovo will be presented in which the GRASP algorithm is applied. Section 6 contains the discussion of the results, the overall performance of the algorithm and its usefulness in light of the assumptions made as well as ideas for further research and development. This is followed by a conclusion summarizing the most relevant findings.

# 2 Literature

As this thesis is mainly concerned with the problem of finding an efficient feasible time-table for the trains specified, first a compact overview of existing literature on the train scheduling problem and proposed solutions for it will be given. First an overview of effective heuristics will be given, then solution techniques based on exact formulations will be discussed. As the goal of this research is to investigate the use of a GRASP, some literature will also be discussed regarding the successful application of this type of optimization scheme. Finally a brief overview of current research on network extending, timetable disturbances and stochastic train delay will be presented.

## 2.1 Heuristics

Over the past few decades, multiple heuristic approaches to different variations of the train scheduling problem have been proposed. Cai et al. (1998) used a greedy algorithm which finds feasible solutions within a short computation time. While this heuristic is designed for single-track railway networks, it could be adapted to also work for mixed single/double-track networks. This heuristic is reported to be able to solve instances up to 100 trains in a reasonable time. A broad overview of heuristics including local search, genetic algorithms and tabu search was presented in Higgins et al. (1997). They conclude that a hybrid algorithm combining a genetic structure and tabu search yields the best results, followed by genetic algorithms, which have a significantly smaller computation time. A side note that needs to be placed is that the generation of the original solution pool is not accounted for in the computation time of the genetic algorithm applied. Again these heuristics are applied to the single-track variant of the problem. The local search approach presented is to select a conflict between two trains that has been resolved by letting one train wait for the other. The algorithm then checks if it is feasible to let the other train wait instead. If this is feasible a new objective value is computed. This idea will form the basis of the adaptive search used in the GRASP in this thesis. Caprara et al. (2006) have used a Lagrangian heuristic to solve large real world instances quickly. While unlike the previous heuristics it can handle double-track sections, a drawback is that this heuristic is only applicable to one-way traffic, and thus solves a severely simplified problem as collisions between trains in opposing directions are not considered. An ant colony inspired heuristic is proposed in Ghoseiri & Morshedsolouk (2005). This is only tested on much smaller instances than relevant for this research. The heuristics mentioned above all focus on speed rather than solution quality. This makes them good candidates to be used in a situation where a large number of solutions is generated and improved upon, like a GRASP framework. While these heuristics have been used and cited often, no literature could be found that uses them in such a context as will be attempted in this research.

## 2.2 MIP formulations and methods

Besides heuristics, many solution techniques have been proposed that formulate the problem in an exact way, which will also be the starting point of this research. In Harbering et al. (2015) a simplified version of the problem is modelled as a job shop problem and solved via dynamic programming. The problem is simplified in that it only accounts for one track networks and assumes equal maximum speed for all trains. While the results look promising, the formulation is only used for relatively small instances and does not easily seem to lend itself to adaptation, making it less suitable for multi track applications. Cacchiani et al. (2008) formulate a model for the train timetabling problem on a one-way corridor and solve it using a column generation approach. While the performance of this approach in terms of computation times is promising, it does not easily lend itself to be extended to heavy traffic two-way corridors. Ghoseiri et al. (2004) constructed a multi-objective formulation of the problem to balance tardiness and fuel consumption. They use a two step solution method, first defining a Pareto frontier, followed by a multi-objective optimization scheme based on this frontier. Castillo et al. (2011) models the problem in a way that is closest to the problem discussed in this thesis, trying out multiple solution techniques on a corridor which is part single-track and part double-track. An algorithm which uses bisection rules to guarantee sharp upper bounds and some variable reduction techniques are applied to keep the algorithm computationally tractable. Their algorithm performs well in a case study, it does however differ from the problem central to this research in that there are no passing loops, hence making overtaking outside of stations not possible. The tackled instances are also smaller than the problems this research aims to tackle. The ideas that are used for the variable reduction form the inspiration for some variable reduction strategies applied in the model constructed in this thesis. In general, the literature is very rich concerning train timetabling for single-track corridors with sidings, and quite some models were also formulated for mixed single and double-track, but relatively limited research has been performed considering mixed track networks with sidings that are subject to heavy traffic in both directions.

## 2.3 Network capacity and bottleneck analysis

Network capacity analysis is a topic that has mostly been studied in the past 15 years. Different approaches have been developed depending on the network structure. A method to estimate the capacity of a double-track network is applied in Landex (2008). They divide the network into sections and use the concept of compressing the timetable to assess the approximate total capacity of a network. Things become slightly more complicated in single-track networks as there is more risk of collision problems. Jamili (2018) propose a similar strategy but tailored to single-track railways. The goal of this research is not necessarily to assess the total capacity of a network, but to quickly and efficiently find potential bottlenecks in the current network. Wahlborg (2004) used a method that uses the current timetable on a network for practical capacity computations. The goal of this research is not necessarily to construct a heuristic for finding the best possible extensions, but to incorporate a method in the GRASP algorithm to indicate potential bottlenecks in the network. These can then be further evaluated and investigated.

## 2.4 GRASP algorithms

A GRASP algorithm is used to find a good feasible solution in a setting where the optimization problem at hand involves a large finite number of alternative choices. The general idea is to iteratively construct solutions in two phases. In the first phase, a greedy algorithm is used to construct a solution, however, there needs to be some form of randomness included in this greedy algorithm. This can be done in a variety of ways. The second stage is to perform a form of local search on the solution found by the greedy algorithm in which the goal is to find the optimal solution in a certain neighbourhood. The best solution found is kept as a reference point and this iterative process is repeated for a certain number of iterations or until a time limit is reached. The term was first used by Feo & Resende (1995). They explain the benefits of this intuitive procedure and use it to solve a few variations on the set covering problem with promising results. Since then the procedure has been used in numerous contexts. Marques-Silva & Sakallah (1999) and Silva & Sakallah (2003) have used the procedure to solve variations of the satisfiability problem (SAT). Vianna & Arroyo (2004) have used it to obtain fast solutions to the multi-objective knapsack problem. Countless other NP-hard optimization problems have been tackled using this framework. Timetabling problems are a notable exception.

## 2.5 Timetable disruptions and stochastic delay

Delays and disruptions occur relatively often in the day to day world of cargo transportation. Ideally, solution methods to timetabling problems should take these events into account so that when they occur the current planning can be simply and efficiently adjusted opposed to creating an entire new timetable from scratch. Liebchen & Stiller (2009) consider the possibility of taking expected delays into account at the moment of scheduling and discuss two heuristics to arrive at a certain level of delay resistance. This is more of an a priori approach to tackling delays. As covering for delays that may not occur leads to loss of efficiency, an additional goal of this research is to find a way to use a GRASP algorithm to repair the timetable quickly once a delay occurs. Albrecht et al. (2013) uses a problem space search heuristic to reschedule trains after disruptions occur. Their idea is to construct a set of decisions from the moment the disruption occurs that lead to a feasible solution. This is repeated in a randomized fashion and the best solution is kept. The suggestions in this thesis are inspired by this idea.

## 2.6 Summary

The exact formulation constructed in this thesis is in part based on the model presented in Castillo et al. (2011). The variable reduction techniques discussed in section 4.2 are inspired by the idea of considering the likeliness of two trains to come into conflict. The approach in this thesis differs from Castillo et al. (2011) in that passing loops and fleet heterogeneity are considered, making the model applicable to more complex networks and train sets. The greedy part of the GRASP discussed in section 4.4 is an extended version of the algorithm proposed in Cai et al. (1998). The main difference is that the greedy algorithm used in this research can handle multiple track. The adaptive search part of the GRASP is inspired by a local search algorithm inHiggins et al. (1997) in which a neighbourhood is defined by saving all conflicts that were resolved in constructing an initial solution. The bottleneck identification process of section 4.5 is inspired by Jamili (2018) in the sense that it investigates which part of the network is busiest given the current timetable. The main difference is that instead of working with a predefined timetable, we use all feasible timetables generated by the GRASP to identify the busiest sections of the network. To my knowledge a GRASP has not yet been used in the current literature to tackle this variant of the time tabling problem. As greedy algorithms and search algorithms exist, and GRASP schemes have been useful for finding solutions to many NP-hard problems, a GRASP scheme for this time tabling problem would be an interesting addition to the current state of the art.

# 3 Problem description and research questions

## 3.1 Problem description

The specific problem at hand is a variation of the train timetabling problem. The network under consideration is a mixed single and double-track network and has passing loops at which trains can pass or overtake each other. The focus will be on a cyclical, freight transportation version of the problem, with the period of a week. The rolling stock will be predefined, meaning that a given set of trains will transport a given set of cargo. Cancellations and back-up trains will not be taken into account and neither will decoupling or other alterations to the fleet. The fleet is not assumed to be heterogeneous. Maximum speed, length and weight may vary among trains. A list of relevant terminology is presented below:

- Station: A station in the network. Trains can wait at station. Some trains may have mandatory stops at some stations. There may be a different number of tracks going into the station and going out of the station.
- Passing loop: A small section of double track that can be used to let a train wait until another train passes.
- Switch: a point in the network where trains can switch track, but which is not a station. There may be a different number of tracks going into a switch and out of a switch and trains can be appointed to any track to continue their journey.
- Decision point: a node in the network at which a decision has to be made that affect the schedule for a train. These consist of all the stations, passing loops and switches in the network.
- Segment: The part of the network between two decision points.
- Track: a single track on a segment. A segment can consist of multiple parallel track.
- Waiting point: The points in the network at which a train can wait to let other trains pass. These consist of the stations and passing loops.
- Capacity: The number of trains that are allowed to wait at a certain waiting point at once. This is generally only one for passing loops.

The goal is to minimize the total travel time of a predetermined set of trains traversing the network. The focus will be on a single mainline corridor in the network, where trains enter the corridor from side branches at their origin station and leave the corridor at their destination. These side branches are not part of the scope and will be disregarded. The trains traversing the corridor all have a set trajectory of stations where they may need to stop. On this corridor no collisions may occur and the stations waiting capacity may not be exceeded during any period in the timetable. The three main decision points in the network are stations where trains can wait (or have to in some cases), switches where trains can switch tracks, and passing loops where trains can wait to let other trains pass or be overtaken. Overtaking is important for efficiency as the trains may have different maximum speeds. Whether a train can switch track or use the passing loop is dependent on a number of attributes including the length and weight of the train. The network is divided up into segments. A segment is defined as the piece of railway between two decision points of the network. The trajectory of a train thus consists of a set of segments and a direction. A solution to this problem consists of the individual timetable and track selections for all trains. The decision variables needed to construct these timetables, are the departure times of the trains at their origin, the track selection on double-track segments for all segments on their trajectory, and the waiting times of the trains at stations and passing loops on their trajectory. If a set of these values presents a solution with no collisions and while respecting all stations waiting capacity on the entire timetable, the solution is feasible.For testing, three networks will be considered consisting of 10, 20 and 30 decision points. The number of trains traversing these networks will vary between 10 and 100. A larger instance based on a real life case is described in more detail in Section 5.4.

## 3.2 Assumptions

**Trains enter the network at stations**

All trains enter the corridor at their origin station. Cai et al. (1998) have shown that this guarantees the existence of a feasible solution without the need for physical backup of trains and they also provide a simple pre-processing method that can be applied in case an instance heavily violates this assumption. As discussed in the introduction the side branches of the network are not taken into account but only the corridor itself to which the entry points are thus regarded as stations.

**No backup**

Trains are not allowed to travel backwards. As the backward speed of most trains is considerably lower than their regular speed, it is unlikely that allowing for backup will improve solution quality. Furthermore, it complicates the problem considerably and as stated in the previous assumption, it is not needed to guarantee feasibility. If physical backup would unexpectedly be needed in a situation, this will need to be manually adjusted by the planner, as it is assumed that in a world without deviations from the timetable it will not be necessary.

**Waiting before entering the corridor**
As the side networks are not part of the problem scope, the assumption is made that before entering the corridor, there is no restriction as to how long a train can wait in the side network. This also implies that the capacity of the side branches is not taken into account.

**No strict time deadlines for departure from origin and arrival at destination**
As the subject of interest is bulk cargo, there is more lenience towards delays and early arrivals than when dealing with passenger trains. The preferred departure time at the origin station of the train is given, but it will be assumed that deviations are allowed under penalisation. As the goal is to maximize the average speed of the trains, additional penalisation for delays is not needed. Penalisation can be added in case the decision is made to depart from the origin station prior to the specified time. This can be done by including early departure in the objective function with a weight coefficient.

**all tracks on a segment have access to the same decision points**
In order to properly define a segment on the network, it will be assumed that all decision points are accessible from all tracks on a segment. That is, if a passing loop is only accessible from one of the tracks on a segment, it will be treated as if it is also accessible from the other track.

**No stopping and constant speed**
It will be assumed that trains travel at a constant speed on network segments and that stopping is not allowed on segments. This assumption should not cause any practical problems, as the speed of trains may still vary on a segment, it will just be assumed in the model that the train travels constantly at its average speed. There is no benefit to stopping on a segment, as passing is not possible there, thus it will always be preferable to stop at a passing loop or station instead. The same argument as before with physical backup applies to practical scenarios where trains must stop on segments.

**Trains can stop at all stations** This implies that if there is a station on the network at which at least a single train must stop, then all trains are allowed to stop at that station. This does not hold true for passing loops, as no train has any a priori required stops at passing loops.

**Stopping requirements are hierarchical**
It is assumed that if a certain train is allowed to stop at a passing loop or station and some other train cannot, then it is assumed that the former can stop at all stations at which the later can stop. This assumption mainly exists to avoid the check for unrealistic conflicts, as will be explained later.

**Single line network**
It will be assumed that the entire corridor under consideration can be viewed as a single line from south to north. There can be multiple tracks parallel to each other and trains travel in one of two directions, north or south on this corridor. This implies that pieces of double-track and passing loops are always accessible from both directions. In Figure 1 one an example of such a representation of a corridor will be given.

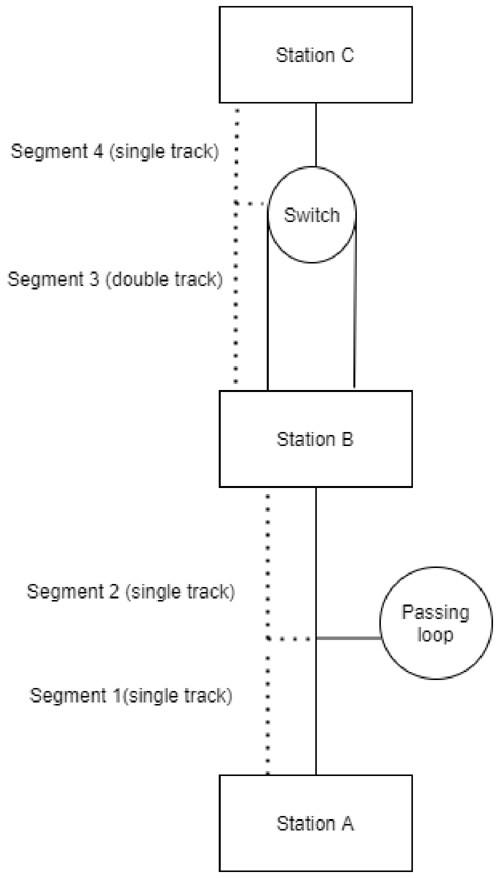

**Figure 1:** A small example of a network with double-tracks and passing loops

## 3.3 Explanation of conflict types

Building an efficient timetable is all about conflict resolution. As the objective is to minimize the total travel time of all trains, a lower bound on the objective is the total travel time of all trains from origin to destination accounting for required stops. This lower bound is strayed from because of conflicts that have to be resolved by trains waiting at stations or passing loops for longer than required. In order to clearly explain how the greedy part of the GRASP deals with conflict resolution, first a clear description of the type of problems that may occur is needed. In this section the main conflict types encountered will be discussed, as well as the precise circumstances in which they occur.

### 3.3.1 Overtaking conflict

Overtaking conflicts occur when a fast train is travelling behind a slow train and at a certain point in the schedule, the fast train will collide with the slow train somewhere in between two waiting points. Specifically, this will happen when a slow train is at its current station, and the location of collision between the two trains lies before the next possible waiting point for the slower train. What complicates the identification of such a conflict is the fact that not all trains can stop at all waiting points. In figure two the possible scenarios in which an overtaking conflict can occur are depicted. Scenario one is the simplest scenario, where the current station at which both trains are residing and the next stop for both is equal. In scenario two the current station for both trains is equal, but the next station for train one is not accessible for train two. Scenario three has train two currently located at a station prior to train one's current station and has the next waiting point after the next station for train one. Scenario four is similar to scenario three but has the second trains next waiting point be the same station as for train one. Scenario five is perhaps the most interesting, where the current station is the same for both trains, but there are one or multiple waiting points for the second train before the collision. In this case, the second train could wait at any of these waiting points to resolve the conflict. Scenario six is similar, but this time the waiting points for the second train lie beyond the conflict location. Note that a lot of other possible configurations that could result in conflict have been left out due to the assumptions in section 3.2, which ensure that the set of stopping points of one train has to be a subset or superset of the other. The conflict is viewed from the perspective of train one. Therefore no stations for train one lying outside its current segment are presented as reaching the next station without any conflict will count as a resolution in this stage. In this representation, it is assumed that train two is the faster train.

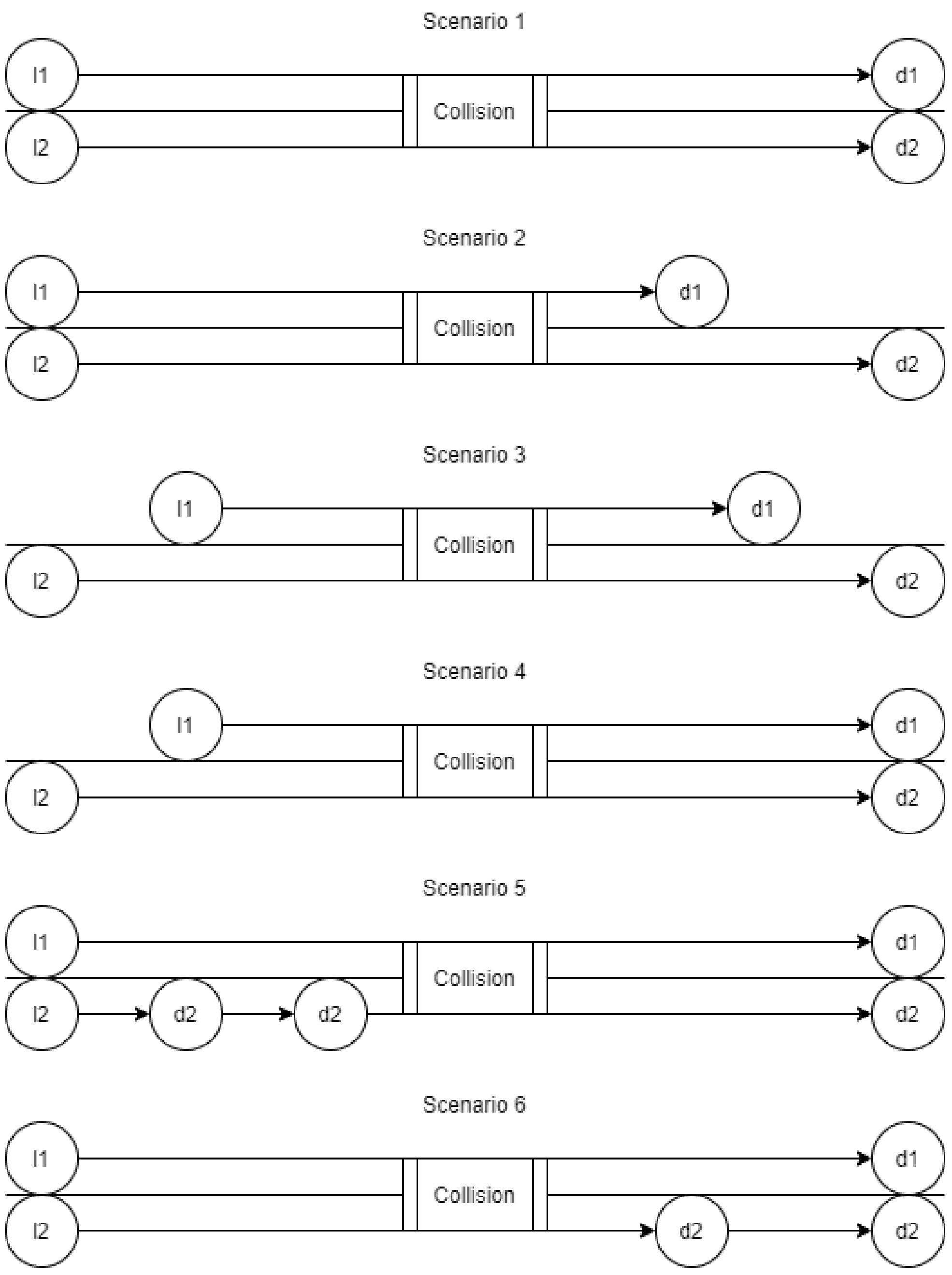

**Figure 2:** The possible scenarios in which an overtaking conflict may occur

### 3.3.2 Collision conflict

Collision conflicts occur when two trains travelling in opposing directions are bound to meet on a segment between two waiting points. Specifically, when two trains depart their current station and one train's next potential waiting point lies beyond the departure point of the other, and their departure happens before the arrival of the opposing train. Figure three shows the possible scenarios in which a collision conflict may occur. The scenarios are comparable to those explained in the section on overtaking conflict, so will not be repeated here.

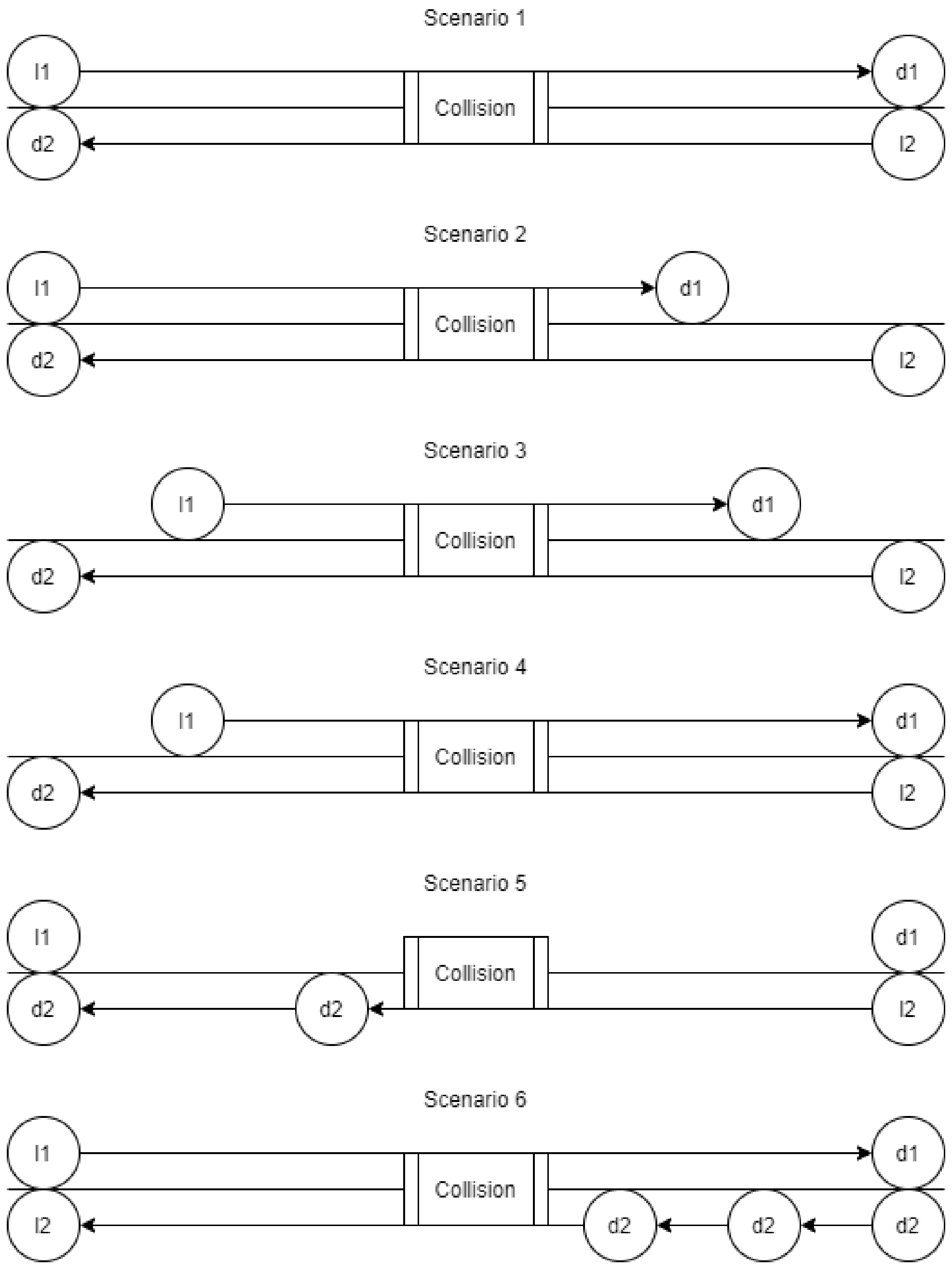

**Figure 3:** The possible scenarios in which a collision conflict may occur

### 3.3.3 Capacity conflicts

A capacity conflict occurs when a train is scheduled to wait at a station during an interval in which at some point there is no capacity for this train to wait there. It is assumed that this does not occur for a train at its arrival or destination station, as by the assumptions of section 3.2 a train can wait just outside of the corridor in the side branch, and once the train arrives at its destination station, it leaves the corridor to another side branch which is not under consideration. For other trains, such a capacity conflict can occur either when it arrives and there is no capacity, or if more trains are scheduled to arrive in the waiting interval than to depart. The solution to such conflict would be to have some trains wait at earlier waiting points until sufficient capacity is available. In figure 4 a visual representation of such a conflict is given at a waiting loop with a capacity of one.

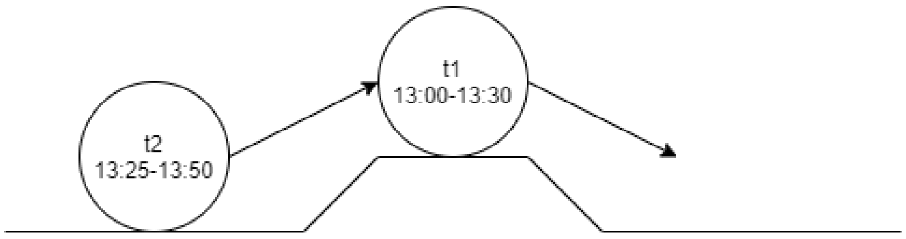

**Figure 4:** An example of a capacity conflict

## 3.4 Research questions

The following questions are to be answered by the research that will be conducted.

- Can an exact model be formulated for the train timetabling problem with single and double-track segments and passing loops that can solve instances up to 50 trains within hours of computation time?
- If the exact model does not allow for adequate computation times for larger instances, can it be used to benchmark a heuristics performance on small instances?
- Can an efficient greedy randomized adaptive search procedure (GRASP) be created for the train timetabling problem with single and double-track segments and passing loops that yields solutions for instances up to 100 trains within hours of computation time with a gap not exceeding 20%?
- How does the GRASP compare to the exact model in terms of solution quality and computation time?
- Can the GRASP also be used to help identify bottlenecks in the current network to aid in decision making for network extensions?
- Can the GRASP be adapted for use in case of maintenance, unexpected alterations on the network and/or delays?

# 4 Methodology

This section starts by defining the problem at hand through the introduction of the relevant parameters and notations. This is followed by a mixed integer program that represents the problem. Variable reduction and bounding techniques are presented in subsections 4.2 and 4.3. Afterwards the GRASP heuristic employed to solve the problem is explained in detail. This is followed by a method to detect potential bottlenecks in a network. Finally in subsection 4.6 some ideas for altering and using the GRASP scheme in case of delays or track maintenance are discussed.

## 4.1 Notation and exact model

Before this section delves into the details of the model, some notation needs to be introduced. Below a quick overview of relevant parameters, sets and decision variables is provided in the notation to be used in modeling.

### 4.1.1 Sets

- T: Trains.
- $T_n$: Trains travelling north.
- $T_s$: Trains travelling south.
- S: Stations.
- P: passing loops.
- K: Switches.
- W: Waiting points, $W = P \cup S$.
- C: Decision points, $C = S \cup P \cup K$.
- $L$ : Segments between decision points.
- $L^r$ : The same set of Segments, but seen from the other direction, so with start and end swapped.
- $R_l$ : The tracks of segment L.

### 4.1.2 Parameters

- $v_t$ : The maximum speed of train t.
- $o_t$ : The origin station of train t.
- $d_t$ : The destination station of train t.
- $cs_t$ : The time it takes for train t to stop when travelling at maximum speed.
- $cd_t$ : The time it takes for train t to reach speed again after stopping.
- $r_t$ : The planned departure time of train t.
- $J_t$ : Trajectory, set of all segments t must travel.
- $I_t$ : Stops, set of all decision points t passes on its trajectory.
- $cap_w$ : Capacity of waiting point w, so the number of trains that can wait there.
- $loc_d$ : The coordinate location of decision point d.
- $start_l$ : Starting decision point of segment l.
- $end_l$ : Ending decision point of segment l.
- $n_l$ : The number of tracks on segment l.
- $g_{t_1,t_2}$ : Binary parameter that indicates whether $t_1$ and $t_2$ are travelling in the same direction.
- $rw_{ts}$ : Required waiting time of train t at station s.
- $rs_{ts}$ : Binary indicating whether train t needs to stop at station s.
- $ao_{tw}$ : Binary indicating whether train t is allowed to stop at waiting point w.
- $M_1, M_2, M_3, M_4, M_5, M_6$ : Big M type constants. Appropriate boundary values discussed in 4.3.
- h: Minimum safety time between two trains going in the same direction. This is also known as headway.
- e: Maximum earliness of departure.
- f: The weight for penalising early departure.
- hor: The planning horizon of the problem instance

### 4.1.3 Decision variables

- $X_t \in [0, e]$ : The early departure of train t as compared to the planned departure time.
- $Y_{tw} \in [0, hor]$ : The waiting time of train t at waiting point w.
- $Z_{tlr} \in \{0, 1\}$ : Binary variable ,indicates whether t uses track r on segment l.

### 4.1.4 Derived variables

These are variables that are not necessarily decision variables but are needed to formulate a complete model. Note that the travel time is in this section as it is intended to have trains travel at the maximum speed that can be achieved while not causing violations.

- $Q_{t_1 t_2 l} \in \{0, 1\}$ : Indicates whether train t1 enters segment l before t2.
- $G_{t_1 t_2 l} \in \{0, 1\}$ : Indicates whether train t1 enters segment l before t2 enters l from the other side.
- $F_{t_1, t_2, d} \in \{0, 1\}$ : Indicates whether train t2 arrives before t1 arrives at decision point d.
- $H_{t_1, t_2, d} \in \{0, 1\}$ : Indicates whether train t2 leaves decision point d before train t1 arrives.
- $W_{tw} \in \{0, 1\}$ : Indicates whether a train waits at a waiting point.
- $A_{td} \in [0, hor]$ : The arrival time of train t at decision point d.
- $D_{td} \in [0, hor]$ : The departure time of train t at decision point d.
- $M_{tl} \in [0, hor]$ : The travel time of train t on segment l.

### 4.1.5 Exact model

$$\min \quad \sum_{t\in T}(r_t - D_{t,o_t}) + f\sum_{t\in T} X_t \tag{1}$$

$$\text{s.t.} \quad D_{t,d} = A_{t,d} + Y_{t,d} \qquad \forall t \in T, c \in I_t \tag{2}$$

$$A_{t,end_l} = D_{t,start_l} + M_{t,l} \qquad \forall t \in T, l \in J_t \tag{3}$$

$$Q_{t_1,t_2,l} + Q_{t_2,t_1,l} = 1 \qquad \forall t1, t2 \in T : g_{t_1,t_2} = 1, l \in J_{t_1} \cap J_{t_2} \tag{4}$$

$$A_{t_1,end_l} + h \leq A_{t_2,end_l} + M_1(3 - Q_{t_1,t_2,l} - Z_{t_1,l,r} - Z_{t_2,l,r}), \qquad \forall t_1, t_2 \in T, l \in J_{t_1} \cap J_{t_2}, r \in R_l \tag{5}$$

$$D_{t_1,start_l} \geq D_{t_2,start_l} - M_2 Q_{t_1,t_2,l} \qquad \forall t1, t2 \in T : g_{t_1,t_2} = 1, l \in J_{t_1} \cap J_{t_2} \tag{6}$$

$$G_{t_1,t_2,l} + G_{t_2,t_1,l^r} = 1 \qquad \forall t1, t2 \in T : g_{t_1,t_2} = 0, l \in J_{t1}, l^r \in J_{t2} \tag{7}$$

$$A_{t_1,end_l} + h \leq D_{t2,start_{l^r}} + M_3(3 - G_{t_1,t_2,l} - Z_{t_1,l,r} - Z_{t_2,l^r,r}) \qquad \forall t1, t2 \in T, l \in J_{t1}, l^r \in J_{t2}, r \in R_l \tag{8}$$

$$D_{t_1,start_l} \geq D_{t_2,start_{l^r}} - M_4 G_{t_1,t_2,l} \qquad \forall t1, t2 \in T : g_{t_1,t_2} = 0, l \in J_{t1}, l^r \in J_{t2} \tag{9}$$

$$M_{t,l} \geq m_l/v_t + W_{t,start_l}(cs_t + cd_t) \qquad \forall t \in T, l \in J_t \tag{10}$$

$$D_{to} = r_t - X_t + Y_{to} \qquad \forall t \in T \tag{11}$$

$$Y_{ts} \geq rw_{t,s} \qquad t \in T, s \in S \tag{12}$$

$$X_t \leq e \qquad \forall t \in T \tag{13}$$

$$Y_{tw} \leq horW_{tw} \qquad \forall t \in T, w \in I_t \tag{14}$$

$$W_{tw} \leq ao_{tw} \qquad \forall t \in T, w \in I_t \tag{15}$$

$$\sum_{t_2\in T: w\in I_{t2}} F_{t_1,t_2,w} - H_{t_1,t_2,w} \leq cap_w - W_{t,w} \qquad \forall t \in T, w \in I_t \tag{16}$$

$$A_{t_1,w} \geq A_{t_2,2} - M_5(1 - F_{t_1,t_2,w}) \qquad \forall t_1, t_2 \in T, w \in I_{t_1} \cap I_{t_2} \tag{17}$$

$$A_{t_1,w} \geq D_{t_2,2} - M_6(1 - H_{t_1,t_2,w}) \qquad \forall t_1, t_2 \in T, w \in I_{t_1} \cap I_{t_2} \tag{18}$$

$$F_{t_1,t_2,w} + F_{t_2,t_1,w} = 1 \qquad \forall t_1, t_2 \in T, w \in I_{t_1} \cap I_{t_2} \tag{19}$$

$$H_{t_1,t_2,w} + H_{t_2,t_1,w} = 1 \qquad \forall t_1, t_2 \in T, w \in I_{t_1} \cap I_{t_2} \tag{20}$$

$$\sum_{r\in R} Z_{t,l,r} = 1 \qquad \forall t \in T, l \in J_t \tag{21}$$

Equation (1) refers to the objective. The goal is to minimize the total travel time of all trains which is represented here by the sum of the arrival time at their destination minus the planned departure time at their origin station. A penalty is added in case a train departs earlier than its originally planned departure time. Equation (2) indicates that the departure time of a train at a decision point should equal its arrival time plus its waiting time. Equation (3) states that the arrival time of any train at the end of a segment is the departure time of that train at the beginning of the same segment plus the travel time on the segment. Equation (4) dictates that for any pair of trains either one enters a segment first. Equation (5) states that if two trains are on the same track for a certain segment, and t1 enters the segment before t2, it has to arrive at the end of the segment before t2. This implies that no collisions happen between trains travelling in the same direction. Equation (6) indicates that if t1 enters a segment before t2, its departure time at the start of that segment should be earlier than the departure time of t2. Equation (7) is the equivalent of equation (4) but for a train travelling in opposite directions. Equation (8) dictates that if t1 enters a segment before t2 from opposing directions, and both are on the same track on the segment, then t2 can only depart the starting point of the segment once t1 has arrived there. This avoids a collision from opposing trains. Equation (9) can be seen as the equivalent of equation (6) but for trains travelling in opposing directions. Equation (10) bound the travel time to the length of the segment divided by the speed of the train, plus the stopping penalty if a train has stopped at the previous station. Equation (11) defines the departure at the origin as the planned departure plus waiting time and the early departure variable. Equation (12) states that the waiting time at a station for a train has to be larger or equal to the required stopping time at a station. Equation (13) bounds the early departure variable by a maximum parameter. Equation (14) states that if a train waits at a waiting point, the number of trains entering the station before this train minus the number of trains leaving the station before this train has to be smaller than the capacity of the waiting point and still leave room for this train to wait. Equation (15) ensures that the waiting time is set to zero if a train does not wait at a waiting point. Equation (16) dictates that a train can only wait at a waiting point if it is allowed to wait there. Equation (17) sets the correct value of the F variables, by stating that if $t_1$ arrives at w before $t_2$, then indeed its arrival time needs to be lower. The same holds for equation (18) but now concerning the H variables and departure time of $t_2$. Equations (19) and (20) can be seen as analogous to equation (4) but now for the F and H variables, ensuring that for a pair of trains only one of the variables attains value one. Finally, equation (21) deals with the track selection, stating that every train can only be on one of the tracks on every segment.

### 4.1.6 Note on circularity

As a sharp eye may have noticed, the model presented above does not explicitly take the circularity of the train schedule into account. This is mainly to avoid adding needlessly complicated notation. However, a feasible repeating schedule can simply be obtained by enforcing constraints 5 to 9 and 16 to 20 between all trains that cross the planning horizon and all those that do not. This would avoid a collision between a train leaving its origin station on Sunday evening and a train departing the opposing station early on Monday morning. A similar tactic will be applied in the GRASP heuristic of section 4.4.

## 4.2 Ideas for variable reduction

The model presented above contains quite a large number of binary variables, making it an unlikely candidate to be useful to solve large real-life instances. Some variable reduction techniques inspired by Castillo et al. (2011) can be used to decrease the number of binary variables used. Two of these ideas were already implemented explicitly in the construction of the model. Firstly, by only using the variables Q for trains going in the same direction and only using variables G for trains travelling in opposing directions, it is ensured that no irrelevant variables are needed to prevent overtaking and collision conflict. Secondly, the variables F and H are only used for train combinations that both visit the station for which the relevant constraint on capacity apply. Another interesting idea is to be more specific about which variables are needed for which combination of trains when it comes to conflicts. Theoretically, every pair of trains going in the same direction could face an overtaking conflict at some point. There are some conditions that if not satisfied make it highly unlikely that this pair of trains encounters a conflict. In a realistic scenario, such a conflict only occurs if the faster train starts its journey behind the slower train and if the two are not so far apart both in time and location that they are likely to meet. Regarding speed, one can simply create the relevant Q variables only for pairs of trains where one is faster than the other. This can either be modelled directly or be done via an intermediate binary parameter that indicates the speed relation between the two trains. To determine if it is likely that the trains meet at all on a certain segment, a function can be constructed using the original departure times, origin, and destination stations of the two types of trains. Let's assume for the sake of simplicity that there is a pair of trains $t_1$ and $t_2$ travelling in the same direction, where $t_2$ is the faster of the two, its expected arrival at the origin station of $t_1$ is greater than the departure time of $t_1$ at that station. A good indication for the likelihood of a collision is to compute the total time that $t_1$ would have to wait longer than $t_2$ to make the two collide. This can then be seen in the light of total travel distance and number of stops of $t_1$. Let

$\pi_{tw}$ denote the expected arrival time of train t at waiting point w. This means that given the current departure times and required waiting times at stations if no additional waiting time was required for conflict resolution, the time that train t would arrive at waiting point w. Similarly, let $\pi_{tx}$ be the expected arrival time of train t at coordinate location x. The relation between these two can be expressed by the following equation:

$$\pi_{tx} = \pi_{tw_x} + |loc_w - x|/v_t$$

The goal is to identify the likelihood that on a segment a situation occurs such that:

$$\pi_{t_1 x} = \pi_{t_2 x}$$

To find this location, both trains need to be observed at the same point in time first. This is done by selecting the train with the earlier departure time, and computing its expected location at the departure time of the later train. This location is calculated as follows:

$$loc_{t_2} = \max_{w \in I_{t_2} : \pi_{t_2,w} < r_{t_1}} loc_w + \max(\pi_{t_2,w} + rw_{t_2 w} - r_{t_1}, 0) v_{t_2}$$

Now that the location of the two trains at the same point in time are computed, the distance between the trains can be taken:

$$D_{t_1,t_2} = loc_{t_2} - loc_{o_{t_1}}$$

And this distance can be transformed into a measure of how much longer $t_1$ would have to wait than $t_2$ for a collision to happen:

$$E_{t_1,t_2} = \frac{loc_{o_{t_1}} + \frac{D_{t_1,t_2}}{v_{t_2} - v_{t_1}} v_{t_1} - \min(loc_{d_{t_1}}, loc_{d_{t_2}})}{v_{t_2}}$$

This number can then be divided by the number of waiting points on the trajectory of $t_1$ to get the additional waiting time per waiting point. Finally, this value can be compared to a threshold parameter, $\gamma$. Such that Q variables are needed only for a pair of trains $t_1$ and $t_2$ if $t_2$ is faster than $t_1$, it starts behind $t_1$ and if the following holds:

$$\frac{E_{t_1,t_2}}{|I_{t_1}|} \leq \gamma$$

This rule would either include all or no Q variables for a pair of trains. To reduce the number of variables even more an alternative can be considered that also takes the segment into account. The new measure of waiting time can be expressed as follows for a pair of trains where $t_1$ is planned to enter segment l before $t_2$:

$$E_{t_1,t_2,l} = \pi_{t_2,start_l} + rw_{t_{start_l}} - \pi_{t_1,end_l}$$

And The Q variables for the pair of trains on this particular segment should only be needed if this value divided by the number of stops $t_1$ has from its origin until segment l is smaller than a value $.Thusweaddthesevariablesifthefollowingholds:$

$$\frac{E_{t_1,t_2,l}}{|I_{t_1,l}|} \leq \gamma$$

A larger value for gamma will lead to a safer solution, with less risk of accidental conflict, but also limits the reduction of variables. Different design choices can be made depending on the desired outcome. Using a relatively low value for $\gamma$ would most likely result in solutions with a few violations. This can be used to find lower bounds for the problem, or be used as a heuristic in combination with a repair mechanic. For the application in this research, a high value of $\gamma$ will be used, as the exact formulation is mostly used to benchmark the performance of the GRASP heuristic discussed later in this text, so any conflicts in a solution are not acceptable. Note that all calculations above are performed for trains heading north. Computations for trains heading south are identical except for some Max statements that have to be Min statements. The selected values for gamma will be discussed in section 5

A similar argument can be made for the reduction of the number of G variables dealing with conflicts between trains travelling in opposing direction. Note that the model already explicitly uses the variables G only if the trains share at least one segment on their trajectory. The idea behind the wait time calculation is different in this case, as letting either one of the trains wait does not necessarily reduce the chance of the trains colliding. Two opposing trains will always interact with one another if they are on the same segment at some point in time. In this light pairs of trains for which this is unlikely to happen can be disregarded due to one train likely passing the other before it even leaves its origin station.

Once again the position of both trains is evaluated at the latest intended departure time among them. Let $t_1$ be the train that departs later. The location of $t_2$ can be computed using the following equation:

$$loc_{t_2} = \max_{w \in I_{t_2}:\pi_{t_2,w} < r_{t_1}} loc_w + \max(\pi_{t_2,w} + rw_{t_2 w} - r_{t_1}, 0)v_{t_2}$$

This time a decision can directly be made based on the location of the second train. The idea is to see how far past the origin station of $t_1$ the second train is as $t_1$ departs it.

$$D_{t_1,t_2} = loc_{t_2} - loc_{o_{t_1}}$$

The extra time that $t_2$ would have to wait longer than $t_1$ in this case is then computed as:

$$E_{t_1,t_2} = \frac{D_{t1,t_2}}{v_{t_2}}$$

Once again this extra waiting time is used to define a condition that needs to be passed in order to construct the relevant G variables:

$$\frac{E_{t_1,t_2}}{|I_{t_2}|} \leq \delta_1$$

Alternatively, it could also happen that the initial distance between two opposing trains is so large, that the train leaving the latest could arrive at its destination before the other train reaches this point in the network. Similar calculations as before can be used. Again let $t_1$ be the train to depart later. The location of $t_1$ at the expected arrival time of $t_2$ at the destination of $t_1$ is needed to continue. All this is under the assumption that the origin of $t_2$ lies further than the destination of $t_1$ otherwise conflict is unavoidable.

$$D_{t_1,t_2} = loc_{d_{t_1}} - (\pi_{t_1,d_{t_1}} - r_{t_2})v_{t_2}$$

Thus the time that $t_1$ must wait longer than $t_2$ is equal to:

$$E_{t_1,t_2} = \frac{D_{t_1,t_2}}{v_{t_2}}$$

A similar condition can be created for this scenario, to only consider the G variables for the pair of trains if:

$$\frac{E_{t_1,t_2}}{|I_{t_1}|} \leq \delta_2$$

Using similar argumentation as for overtaking conflicts, the decision to include the variables in the model can be made on a segment level as well, leading to the following decision rule for a pair of trains for which $t_1$ is bound to enter segment l before $t_2$:

$$E_{t_1,t_2,l} = \pi_{t_1,end_l} - (\pi_{t_2,end_l} + rw_{t_2,end_l})$$

$$\frac{E_{t_1,t_2,l}}{|I_{t_1}|} \leq \delta$$

For the values of $\delta$ similar reasoning holds as for the value of $\gamma$. In this research again high values will be used to ensure feasible solutions at all times. More details about selected values will be given in section 5.

## 4.3 Appropriate values for big M constants and upper bounds on non-binary variables

To efficiently define the model good boundary values for the big Ms used in some of the constraints must be set. Theoretically the differences in arrivals and departures between trains can span the entire planning horizon. This means that a tighter bound than using the planning horizon for the big M parameters is not possible. Fortunately, there is something else we can do to improve the bounds related to these constraints. The goal of tightly bound M parameters is to give good implicit bounds for the A, Y, M and D variables, which are the only non-binary variables that appear in the relevant constraints. Instead of using a stricter M to bound these variables, a reasonable restriction can be made that will allow for a reduction of the solution space that is unlikely to exclude the optimal solution. Starting with the M variables resembling travel time on a segment. In the assumptions of section 3.2 it is assumed constant speed on a segment and no stopping on a segment. The combination of these assumptions makes it implicit that the travel time on a segment for a train in the optimal solution should be equal to the length of the segment divided by the maximum allowed speed on the segment. Thus to get a tighter bound of the M variables the greater equal sign in constraint 10 can be replaced by an equality sign. For the arrival and departure times of the trains, a boundary condition based on the expected arrival and departure times can be constructed:

$$UpperBoundA_{t,w} = \eta(\pi_{t,w} - r_t) + r_t + \epsilon$$

$$UpperBoundD_{t,w} = \eta(\pi_{t,w} + rw_{t,w} - r_t) + rt + \epsilon$$

These restrictions state that the arrival/departure time of a train at a waiting point must be within a factor $\eta$ of its expected arrival/departure time plus some lenience parameter epsilon. Finally, the waiting time at any waiting point can be bound to the total expected travel time of the train on the network:

$$UpperBoundY_{t,w} = \pi_{t,d_t} - r_t + \epsilon$$

Note that the reduction suggested in section 4.2 is in essence a relaxation, as some variables are omitted from the model along with their relevant constraints. The bounds specified in this subsection constitute a restriction on the model. The goal is to choose the $\gamma, \delta$, and $\eta$ parameters in such a way that the objective found remains the same while lowering the computation time needed. A sensitivity analysis on the values of these parameters and corresponding results are discussed in section 5.

## 4.4 GRASP

In this section the proposed greedy randomized adaptive search procedure to solve the timetabling problem will be discussed. First, the greedy part is explained, followed by a thorough explanation of the adaptive search and a discussion on the neighbourhood used. Finally, the whole algorithm is laid out and some notes are given on the meta-parameters used.

### 4.4.1 Greedy Heuristic

The greedy part of this GRASP is inspired by and extended from Cai et al. (1998). The main differences that need to be addressed are the following: the problem solved in Cai et al. (1998) does not allow for sections with multiple tracks in the network. It also does not consider the possibility that not all trains can stop at all stations or waiting points. These two additions make the problem a lot more complex and thus a lot needs to be changed to get a functional greedy heuristic. Like in Cai et al. (1998), the algorithm will be divided up into routines that can be called whenever a problem is encountered. The idea of a location time pair for every train in the network is also adapted, but instead of saving the location, only the current station is used. Before the details of the GRASP are discussed, some additional notation is needed. The notation not explicitly mentioned here is identical to the notation discussed in section 4.1. Note that in this entire section, all computations will be performed under the implicit assumption that $t_1$ is northbound whenever mentioned. To get the correct computations for southbound trains, the exact same logic can be followed, but as for southbound train travelling further leads to a lower value for their current location, some pluses need to be exchanged for minuses. If this arises in non-trivial situations it will be explicitly addressed, but for simple computations, this will not be repeated for the sake of brevity.

- $\chi_t$ : The current station time pair for a train, representing its current station and the departure time at that station.
- $\tau_t$ : The current time of train t in the current state of the algorithm
- $\sigma_t$ : The current station t is residing at in the current state of the algorithm
- $\lambda_t$ : The current location of the station t is residing at in the current state of the algorithm
- $track_{l,t}$ : The track used by train t on segment l in the current state of the algorithm.

- $next_t$ : The waiting point that t is scheduled to visit next in the current state of the algorithm.
- $previous_t$ : The waiting point that t has visited right before getting to the current one.
- $\eta_{t,l}$ : Indicates whether t has already switched tracks on this segment during this iteration.
- $\rho_{t,w}$ : The expected departure of train t at waiting point w given the current state of the algorithm. This is computed as expected arrival time $\pi_{t,w}$ plus the waiting time $Y_{t,w}$

As explained in the assumptions of section 3.2 the initial time location pair for each train consists of their planned departure time, and origin station. In addition, all trains are set to use solely the main track on the entire network.

```
for $t \in T$ do
    $\chi_t = \{o_t, r_t\}$
    for $j \in J_t$ do
        $Z_{tl1} = 1$ $Z_{tl2} = 0$
    end
    for $w \in I_t$ do
        $Y_{tw} = rw_{tw}$
    end
end
```

**Algorithm 1:** Initialisation

Now the different conflict types will be discussed. For the conflicts described in section 3.3, a procedure will be introduced to detect them, and a routine will be constructed to solve them.

A train $t_1$ is bound to be in an overtaking conflict with $t_2$ on the next segment it will traverse, given the current state of the algorithm, if the following conditions hold.

- $t_2$ is travelling in the same direction as $t_1$.
- $t_2$ has a greater maximum speed than $t_1$.
- $t_2$s next waiting point lies beyond $t_1$s current location.
- The expected departure of $t_2$ at $t_1$s current station is later than the expected departure of $t_1$.
- The expected arrival of $t_2$ at the next waiting point for $t_1$ is earlier than the expected arrival of $t_1$.

- On the the point of collision, $t_1$ and $t_2$ are using the same track.

Most of these can be checked quite easily, the exception being the last condition, as computing the collision location is not trivial due to the fact that there may be multiple waiting points on the way for train two. Before continuing the method of finding the collision location will be explained. The following computations are done under the assumptions that all other checks are passed. First both trains need to be evaluated at the same point in time. Therefore the location of $t_1$ is considered at the moment $t_2$ is bound to depart from $t_1$'s current location.

$$loc_{t_1} = \lambda_{t_1} + (\pi_{t_2,\sigma_{t_1}} + Y_{t_2,\sigma_{t_1}} - \tau_{t_2})v_{t_1}$$

Now the distance between the two trains is

$$D = loc_{t_1} - \lambda_{t_1}$$

It would seem that the collision location can be computed easily by

$$col = \lambda_{t_1} + \frac{D}{v_{t_2} - v_{t_1}} v_{t_2}$$

and this would be true, as long as there are no potential waiting points in between $\lambda_{t_1}$ and the computed collision point. If there are any, the waiting time of $t_2$ at these waiting points needs to be taken into account. This is done iteratively:

```
start = $\lambda_{t_1}$, end = col
$wait = \sum_{w \in W: start < loc_w < end} Y_{t_2 w} + W_{t,w}(cs_t + cd_t)$
Last = the last waiting point of $t_2$ before the collision
while $wait > 0$ do
    $loc_{t_1} = loc_{t_1} + wait v_{t_1}$
    $D = loc_{t_1} - \lambda_{t_1}$
    $col = \lambda_{t_1} + \frac{D}{v_{t_2} - v_{t_1}} v_{t_2}$
    start = $loc_{Last}$
    end = col
    $wait = \sum_{w \in W: start < loc_w < end} Y_{t_2 w} + W_{t,w}(cs_t + cd_t)$
    Last = the last waiting point of $t_2$ before the collision
end
return col
```

**Algorithm 2:** Algorithm to compute collision point

This will give us the actual collision location. Now a check needs to be performed to verify whether the same track at the collision location. by checking that $track_{l,t_1} = track_{l,t_2}$

The total checking algorithm for $\hat{t}$ can thus be defined as follows

```
for $t \in T : g_{t,\hat{t}} = 1, v_{\hat{t}} > v_t$ do
    if $loc_{next_t} > \lambda_{\hat{t}}$ & $\pi_{t,\sigma_{\hat{t}}} + Y_{t,\sigma_{\hat{t}}} > \tau_{t_1}$ & $\pi_{t_2,next_{\hat{t}}} < \pi_{\hat{t},next_{\hat{t}}}$ then
        loc = collision location between t and $\hat{t}$
        seg = segment on which loc lies.
        if $track_{seg,t} = track_{seg,\hat{t}}$ then
            return true, t, seg;
        end
    end
end
return false;
```

**Algorithm 3:** Subroutine to check for overtaking conflicts

When a conflicting train has been identified, the conflict needs to be resolved. This can be done in four possible ways. Either the slow train or the fast train changes track, or one of the trains waits at a waiting point. This will be the first time that randomness comes into play, as which resolution method is selected is determined by a set of probability parameters.

- $\omega_{fastTrack}$ : The probability that the resolution method in which the fast train switches track is selected.
- $\omega_{slowTrack}$ : The probability that the resolution method in which the slow train switches track is selected.
- $\omega_{fastWait}$ : The probability that the fast train waits to resolve the conflict.
- $\omega_{slowWait}$ : The probability that the slow train waits to resolve the conflict.

Track switching needs to be handled with care, as switching the same train back and forth may result in an infinite loop of different conflicts. Because of this, a train will only be allowed to switch track on every segment once during one iteration of the greedy algorithm. For this $\eta_{t,l}$ is used. The same problem occurs when the train that does not switch tracks now encounters another conflict on the same segment. Therefore a conflict can only be resolved by track switching if neither of the involved trains has switched tracks before on this segment. In addition, track switching is only possible if there are multiple tracks on the segment. When letting the fast train wait, there may be multiple options for a decision point to wait at. Due to the assumptions of section 3.2 it is known that except for the current decision point all of these are passing loops. As capacity is scarce on passing loops, the decision is made to let the fast train wait at its current station rather than one of the other options. The following subroutine is used to solve an identified overtaking conflict, where $t_1$ is the slow train, $t_2$ is the fast train, seg is the segment on which the conflict occurs.

```
rnd = random number between 0 and 1
if $n_{seg} > 1$ & $\eta_{t_1,seg} = \eta_{t_2,seg} = 0$ & $rnd < (\omega_{slowTrack} + \omega_{fastTrack})$ then
    rnd = random number between 0 and 1
    if $rnd < \omega_{slowTrack}$ then
        $track_{seg,t_1} = 2$
        $\eta_{t_1,seg} = \eta_{t_2,seg} = true$
    end
    else
        $track_{seg,t_2} = 2$
        $\eta_{t_1,seg} = \eta_{t_2,seg} = true$
    end
end
else
    $rnd = rdn(\omega_{fastWait} + \omega_{slowWait})$
    if $rnd < \omega_{slowWait}$ then
        $D_{t_1,\sigma_{t_1}} = \pi_{t_2,\sigma_{t_1}} + Y_{t,\sigma_{t_1}} + h$
        $Y_{t_1,\sigma_{t_1}} = D_{t_1,\sigma_{t_1}} - A_{t_1,\sigma_{t_1}}$
        $\tau_{t_1} = D_{t_1,\sigma_{t_1}}$
    end
    else
        $wait = \pi_{t_1,next} - \pi_{t_2,next_{t_1}}$
        $D_{t_2,\sigma_{t_2}} = D_{t_2,\sigma_{t_2}} + wait + h$
        $Y_{t_2,\sigma_{t_2}} = D_{t_2,\sigma_{t_2}} - A_{t_2,\sigma_{t_2}}$
        $\tau_{t_2} = D_{t_2,\sigma_{t_2}}$
    end
end
```

**Algorithm 4:** Subroutine to solve overtaking conflict

Next, an algorithm will be explained for dealing with capacity issues. For all trains a check is performed to indicate whether the station or loop they are currently waiting at has enough capacity to harbour them the entire waiting interval. The idea is to hold back the train at its previous station for a time until sufficient capacity is available. The minimum required waiting time at the station needs to be taken into account as well. The new arrival time of this train at this station should be so that the train can wait until at least its current departure time at the station, and wait for at least its minimum required waiting time. The capacity of a station at time i can be defined as:

$$cap_{w,i} = cap_w - |\{t \in T : A_{t,w} < i \wedge D_{t,w} > i\}|$$

Suppose a train is set to wait at waiting point w, for the interval [a,b]. The first step would be to determine the capacity at time b. If this is insufficient already, the whole waiting period will need to be moved to the previous station. If the capacity is sufficient, the last arrival or departure before b is selected. In case of a departure, the capacity prior to this event is one less. In case of an arrival, it is one more. The bottleneck point is moved back until either a point at which there is no capacity for this train to wait or point a is reached. If point a is found, there is no capacity conflict. If at moment c there is insufficient capacity for the last time, either the following holds:

$$c - b \geq r_{t,w}$$

in which case the train is backed up and left to wait at the previous waiting point. The waiting time will be set such that the train will arrive at its next station at time c and depart at time b. This covers the minimum required waiting time. If this does not hold, look ahead of time b for $r_{t,w} - (b - c)$ and check if there still remains capacity for this train to wait. If yes, it should be scheduled to arrive at w at time c, and let it wait there until $c + r_{t,w}$. If there is no capacity, the required waiting time will be dealt with at a later stage. For now, let the entire interval be waited at the previous station, so that for now the conflict is resolved. The same train may encounter this conflict again if the state of the algorithm changes but this is not a problem. The subroutine for checking a train for a capacity conflict and solving it if found is:

```
$bottleneck = cap_w - cap_{w,b}$;
$time = b$
while $bottleneck > 0 \wedge time > a$ do
    $LastDeparture = \max_{t \in T : D_{t,w} < b} D_{t,w}$
    $LastArrival = \max_{t \in T : A_{t,w} < b} A_{t,w}$
    if $LastDeparture > LastArrival$ then
        $bottleneck - -$
        $time = lastDeparture$
    end
    else
        $bottleneck + +$
        $time = lastArrival$
    end
end
if $time < a$ then
    return false // no conlfict
end
if $b - time > r_{w,t} \vee$ sufficient capacity on interval $[b, time + r_{w,t}]$ then
    $D_{t,previous_t} = D_{t,previous_t} + (time - A_{t,w})$
    $Y_{t,previous_t} = D_{t,previous_t} - A_{t,previous_t}$
    $Y_{t,w} = \max\{D_{t,w} - time, r_{t,w}\}$
    $\sigma_t = previous_t$
    $\lambda_t = loc_{previous_t}$
    $\tau_t = D_{t,\sigma_t}$
end
else
    $D_{t,previous_t} = D_{t,previous_t} + (b - A_{t,w})$
    $Y_{t,previous_t} = D_{t,previous_t} - A_{t,previous_t}$
    $Y_{t,w} = r_{t,w}$
    $\sigma_t = previous_t$
    $\lambda_t = loc_{previous_t}$
    $\tau_t = D_{t,\sigma_t}$
end
```

**Algorithm 5:** subroutine to check for and solve capacity conflict

The above algorithm for resolving capacity conflicts can also be used to fix the situation where a train is scheduled to wait at a decision point at which it is not allowed to wait (a switch, or a passing loop for which the train is too long or too heavy). This decision point will then be appointed a waiting capacity of zero for the corresponding train and the procedure sets the train back to a station at which it can wait.

Finally, the last major conflict type to be discussed are collision conflicts between trains travelling in opposite directions. As with overtaking conflicts, a list will be made of conditions that indicate a conflict. The algorithm for computing the collision locations is discussed, and afterwards, the algorithm for finding and resolving the conflict is provided. Northbound train $t_1$ is bound to be in a conflict with Southbound train $t_2$ on the next segment traversed by $t_1$ if the following conditions are met.

- $t_2$ is travelling in the opposite direction of $t_1$
- $t_2$ is currently at a point in the network where it has yet to encounter $t_1$
- the expected departure time of $t_2$ at the next waiting point for $t_1$ is earlier than the expected arrival of $t_1$
- the expected arrival of $t_2$ at the current station of $t_1$ is later than $t_1$'s departure
- at the location of collision both trains are using the same track

To compute the collision location, assuming all other conditions have been met, both trains need to be considered at the same time. The train that enters the segment under consideration last will be taken as reference. If $t_1$ departs earlier from its current station than $t_2$ from the next waiting point of $t_1$, the location of $t_1$ at the moment that $t_2$ departs is considered, and vice versa.

```
$lastDeparture = \arg\max_{w \in I_{t_2}: \rho_{t_2,w} < \pi_{t_1,w}} \rho_{t_2,w}$
$lastArrival = \arg\max_{w \in I_{t_2}: \pi_{t_2,w} < \pi_{t_1,w}} \pi_{t_2,w}$
if $lastDeparture = lastArrival$ then
    if $\rho_{t_2,lastDeparture} > \tau_{t_1}$ then
        $loc_{t_1} = \lambda_{t_1} + (\rho_{t_2,lastDeparture} - tau_{t_1})v_{t_1}$
        $loc_{t_2} = loc_{lastDeparture}$
        $D = loc_{t_2} - loc_{t_1}$ $col = loc_{t_1} + \frac{D}{v_{t_1}+v_{t_2}} v_{t_1}$
    end
    else
        $loc_{t_2} = loc_{lastDeparture} + (\tau_{t_1} - \rho_{t_2,lastDeparture})v_{t_2}$
        $loc_{t_1} = \lambda_{t_1}$
        $D = loc_{t_2} - loc_{t_1}$ $col = loc_{t_1} + \frac{D}{v_{t_1}+v_{t_2}} v_{t_1}$
    end
    return col, this is the collision location.
end
else
    return false, the meet happens as $t_2$ is waiting at a loop or station
end
```
**Algorithm 6:** Algorithm for finding the collision location between two opposing trains

The algorithm to check for a collision conflict

```
for $t \in T : g_{t,\hat{t}} = 0, \lambda_t > \lambda\hat{t}$ do
    if $loc_{next_t} > \lambda_{\hat{t}}$ & $\rho_{t,next_{\hat{t}}} < \pi_{t_1,next_{t_1}}$ & $\pi_{t_2,\sigma_{\hat{t}}} > \tau_{t_1}$ then
        loc = collision location between t and $\hat{t}$
        seg = segment on which loc lies.
        if $track_{seg,t} = track_{seg,\hat{t}}$ then
            return true, t, seg;
        end
    end
end
return false;
```
**Algorithm 7:** The subroutine to check for a collision conflict

There are once again four options to resolve the conflict the $\kappa$ parameters represent the probabilities for changing track. Contrary to letting either train wait according to some fixed probability as in the overtaking conflict resolution, the probability that a train will wait instead of the other is now based on which train needs to wait longer. In overtaking conflicts it is usually locally optimal to let the slow train wait, but in collision conflicts, this is not applicable, so the additional waiting time for both scenarios needs to be evaluated.

```
rnd = random number between 0 and 1
if $n_{seg} > 1$ & $\eta_{t_1,seg} = \eta_{t_2,seg} = 0$ & $rnd < (\kappa_{northTrack} + \kappa_{southTrack})$ then
    if $rnd < \kappa_{northTrack}$ then
        $track_{seg,t_1} = 2$
        $\eta_{t_1,seg} = \eta_{t_2,seg} = true$
    end
    else
        $track_{seg,t_2} = 2$
        $\eta_{t_1,seg} = \eta_{t_2,seg} = true$
    end
end
else
    $stat = \arg\min_{w \in I_{t_2}:loc_w > col} loc_w$
    $wait1 = \pi_{t_2,\sigma_{t_1}} - D_{t_1,\sigma_{t_1}}$
    $wait2 = \pi_{t_1,stat} - \rho_{t_2,stat}$
    $frac = \frac{wait1}{wait1+wait2}$
    if $rnd < frac$ then
        $D_{t_1,\sigma_{t_1}} = \pi_{t_2,\sigma_{t_1}}$
        $Y_{t_1,\sigma_{t_1}} = D_{t_1,\sigma_{t_1}} - A_{t_1,\sigma_{t_1}}$
        $\tau_{t_1} = D_{t_1,\sigma_{t_1}}$
    end
    else
        $Y_{t_2,stat} = Y_{t_2,stat} + wait2$
        if $stat = \sigma_{t_2}$ then
            $D_{t_2,stat} = Y_{t_2,stat} + A_{t_2,stat}$
            $\tau_{t_2} = D_{t_2,stat}$
        end
    end
end
```

**Algorithm 8:** Subroutine to solve a collision conflict

Before giving the total outline of the greedy heuristic, the pseudo code for moving a train to the next station should be discussed. The following updates need to be made when moving a train to its next waiting point.

```
$A_{t,next_t} = \pi_{t,next_t}$
$D_{t,next_t} = A_{t,next_t} + Y_{t,next_t}$
$\sigma_t = next_t$
$\tau_t = D_{t,\sigma_t}$
$\lambda_t = loc_{\sigma_t}$
```

**Algorithm 9:** procedure to move a train to the next decision point

The total greedy heuristic can now be defined:

```
Step 0: Perform initialisation
while $\exists t \in T : \sigma_t \neq d_t$ do
    Step 1:
    for $t \in T : \sigma_t \neq d_t$ do
        Use Algorithm 1 to check for overtaking conflict if conflict found then
            Solve conflict using algorithm 2,
            Go to step 1.
        end
        Use Algorithm 3 to check for and resolve a capacity conflict and illegal
          waiting
        if conflict found then
            Go to step 1.
        end
    end
    Step 2:
    Set the train with the lowest current time to be incumbent.
    Check whether the incumbent train has any collision conflicts using algorithm 4
    if conflict found then
        Solve the conflict using algorithm 5 Go to step 1.
    end
    else
        Select the train with the lowest current time as incumbent. Move incumbent
          train to next decision point.
    end
end
```

**Algorithm 10:** The complete greedy heuristic

### 4.4.2 Adaptive Search

The adaptive search part of the GRASP scheme will be based around the encountered conflicts in the greedy part. Every conflict that is encountered will be recorded along with its time of occurrence and its method of resolution. The neighbourhoods suggested in Higgins et al. (1997) and Albrecht et al. (2013) change the resolution method of one or more encountered conflicts, check if the problem is still feasible, and if so, evaluate the new objective. As changing a track assignment in a solution will be very likely to lead to many new conflicts and infeasibilities, the track assignment will not be a part of the adaptive search explained here but will be fully determined by the greedy heuristic. The adaptive search takes a conflict, changes its resolution method and from that point in time tries to construct a solution that is feasible and as close as possible to the starting solution. Every time a conflict is encountered between two trains, the resolution method chosen will be the one also used in the starting problem. If a new conflict is encountered that did not occur in the starting problem, the same strategy as in the greedy heuristic will be applied to obtain a feasible solution. Two approaches can be investigated. The first is to accept an altered resolution as soon as its resulting objective is better than the current objective, the second is to check all resolution swaps and choose the best one. For smaller instances, the second will lead to better quality solutions, but as it is much more computationally expensive, it is not suited to be used for larger instances. Let C be the set of all conflicts encountered in the greedy heuristic. High-level pseudo-code for a single iteration of the two adaptive search variations is given below:

```
for $c \in C$ do
    Copy the starting solution up till the occurrence of conflict c
    Finish the timetable, using the same resolution methods where possible
    Obtain the new objective
    rnd= random number between 0 and 1
    if new objective<current objective XOR rnd¡SimulatedAnnealingProb then
        Replace current solution by adaptive search solution. Break loop;
    end
    Stop local search //a local optimum is found
end
```

**Algorithm 11:** adaptive search where the first improvement is selected

```
for $c \in C$ do
    Copy the starting solution up till the occurrence of conflict c
    Finish the timetable, using the same resolution methods where possible
    Obtain the new objective
    rnd= random number between 0 and 1
    if new objective¡current best search objective then
        Replace current best search solution by current adaptive search solution.
    end
end
if best search objective ¡ current objective XOR rnd¡SimulatedAnnealingProb then

end
Replace current solution with best search solution  else
    Stop local search // a local optimum is found
end
```

**Algorithm 12:** adaptive search where the best improvement is selected

To prevent getting stuck in a local optimum too often, a simplified version of simulated annealing can be applied. This will allow for the possibility of accepting a slight increase in the objective value.

### 4.4.3 Algorithm structure and meta-parameters

Now with greedy heuristic and the adaptive search defined, a full overview of the GRASP scheme can be given. In the description below, a solution is made up of all arrival and departure times of the train, the actual timetable so to speak, as well as the encountered conflicts and the chosen resolution methods.

```
BestObjective= $\infty$
BestSolution=null
for i=0 to GreedyIterations do
    Run Algorithm 6 to get a current solution and objective
    for j=0 to SearchIterations do
        Run Algorithm 7 or 8 to improve on the current solution and save the best
          found to current
    end
    if currentObjective¡BestObjective then
        Set BestObjective to currentObjective and BestSolution to CurrentSolution.
    end
end
```

**Algorithm 13:** GRASP

An important design choice is deciding on appropriate values for "GreedyIterations" and "SearchIterations". If one uses the first improvement search tactic, a larger number of iterations can be used as the computation time will be lower. As the track selection is specified entirely by the greedy heuristic, the number of greedy iterations should not be too low in favour of excessive searching, as this could increase the risk of getting stuck in local optima. The probabilistic parameters that influence the resolution methods used should also be set sensibly. It seems logical to assign equal probability to either train changing track when possible. The total probability for track switching should not be set too high, as trains may encounter multiple conflicts on a single segment, and tracks can only be changed once. It should also not be set too low, as changing track is a "free" resolution, meaning it does not increase the overall waiting time. In case of overtaking conflicts, it seems logical to assign a greater probability to the scenario where the slow train waits for the fast train, as letting the fast train wait is likely to lead to another conflict between the two trains at later segments. The last parameter to be discussed is the simulated annealing parameter used in the local search scheme. The value of this probability should be according to the number of iterations. If rather few iterations are executed the value should be kept very low, as discarding good solutions makes it unlikely for them to be encountered again. If a very large number of iterations is performed a larger value can be used, as it leads to exploring more regions of the neighbourhood. Different parameter values will be tested and discussed in section 5.

## 4.5 Bottleneck identification

As the GRASP algorithm is likely to traverse many feasible solutions, a nice opportunity presents itself to identify potential bottlenecks on the network, for the current planning. For every feasible solution, one can count for each leg how many conflicts occur on it. After running the GRASP, the average number of conflicts for every segment on the network can be evaluated, as well as the standard deviation of the number of conflicts on this segment. The average additional waiting time per conflict on the segment can also be easily obtained. These three statistics can be used as indicators to bottlenecks in the network that could benefit the most from a piece of additional track, or perhaps one or more extra passing loops. Another option is to weigh these statistics by the quality of solutions. An argument for doing so it that it will show you how very good solutions can be made even better, an argument against could be that when there are slight changes in planning an extension that is specified too much on very good solutions may have less added value than an extension that could help improve lower-quality solutions. The statistics recorded can be transformed into a score for a segment, which shall be named the bottleneck factor. Let $\bar{X}_l$ be the average number of incidents on segment l over all found feasible solutions, and let $\S_l$ be the standard deviation of the number of accidents and $\bar{D}_l$ the average delay. A bottleneck factor can be computed as:

$$B_l = \frac{\bar{X}_l \bar{D}_l}{S_l}$$

The reason a larger standard deviation in the number of conflicts decreases the bottleneck factor is that it indicates that over many solutions the number of conflicts is unreliable, and may be larger or smaller, so the extension may be less useful in many scenarios. Let $z^*$ be the best found objective value and $z_s$ be the objective value corresponding to a feasible solution s. Let $\bar{D}_{l,s}$ be the average increase in waiting time per conflicts on segment l in solution s and $X_{l,s}$ the number of conflicts on segment l in solution s. A weighted bottleneck factor can be computed as follows:

$$\hat{B}_l = \frac{X_{l,s} \bar{D}_{l,s}}{z_s / z^*}$$

Note that these two factors are not on the same scale and thus for decision support one should compare two solutions using the same factor for both.

## 4.6 Adaptation to delays and disruptions

In this section, four different disruption types will be discussed, as well as how the greedy algorithm can be used to deal with them.

### 4.6.1 Planned track maintenance

Suppose during some interval on the timetable a certain track needs maintenance and this is known in advance. If the piece of track that needs maintenance is on the main track at a point where there is no double-track, all trains scheduled to enter that segment after maintenance starts must wait at their previous waiting points until the point where they are bound to enter the segment after maintenance is done. Any capacity conflicts arising from this process can be handled by algorithm 3. If the maintenance is performed on one track of a double-track segment, all trains are scheduled to use the other track. This is achieved by not allowing track switching as a resolution method in algorithms 2 and 5.

### 4.6.2 Unplanned track maintenance

Suppose that while using the timetable a track breaks down. The greedy heuristic provided can be used to efficiently find a new feasible solution. If the track in question is the only track on the segment, all trains bound to enter the segment need to wait at their last possible waiting point until after repairs are done. This is similar to how any a priori known repairs would be handled. Any capacity conflicts can once again be dealt with by using algorithm 3. If the broken track is on a double-track segment, all trains need to be rescheduled on the other track until repairs are done. This will likely result in some new collision and overtaking conflicts. These can then be handled by Algorithms 2 and 5. There are two options, either just use the greedy algorithm to complete the planning from the disruption onward, or use the same idea as in the adaptive search, as to only make changes when the current timetable is infeasible. Both would yield feasible solutions quickly and one does not necessarily perform better in terms of objective, though from a communication and planning point of view, preserving as much of the original timetable as possible may be preferred.

### 4.6.3 Delay robust timetable

Delays in the schedule may occur due to all sorts of circumstances. These delays may cause new conflicts to occur, but may also occasionally prevent some other conflicts from happening. Historical data on train delays can be used to estimate a distribution for the percentage of trains that are delayed and their mean delay time. An intuitive approach to robustness against delays of this type, is to change the way collision and overtaking conflicts are detected. In the greedy algorithm, it is assumed the travel times on a segment are fixed. However, these methods could be altered to also detect a conflict given the possibility of delay. Suppose that from historical data it is known that $\psi$ per cent of trains encounters a delay, with an average delay time of $\bar{D}$. Let n be the number of segments train t still needs to pass before arriving at w given its current station. In conflict detecting, all expected arrival times can be transformed into intervals as follows:

$$\pi_{t,w} \rightarrow \Pi_{t,w} = \{\pi_{t,w}, \pi_{t,w} + n\psi\bar{D}\}$$

and adapt the additional waiting time for the train selected to wait so that given any time on this interval no collision occurs. This will of course lead to a less efficient schedule, but it will be more robust to expected delays. The downside of such an approach is that delays are not likely to be normally distributed, but follows more heavy-tailed distribution forms as discussed in Yang et al. (2019). This means that this robustness might not hold very well if the delay times vary a lot and may make the schedule needlessly inefficient. One might therefore consider fixing problems caused by delays in an a postiori manner.

### 4.6.4 Dealing with unexpected delays

Before altering the timetable, it needs to be assessed whether the delay in question causes any new conflicts or changes the nature of any resolved conflicts. If it is possible for the train in question to just subtract the delay from its waiting time at following waiting points, whilst still meeting the required waiting times, and not altering any conflicts until it does so no alterations to the timetable need to be made. This condition is easy to check by just following the original timetable and inspecting the conflicts this train encounters before the opportunity to compensate for the delay is given. If the delay causes new conflicts or alters the resolution of existing conflicts, the simplest way to get back to a feasible timetable is to fix all trains current time and current position to their latest before the delay occurred, and use the greedy algorithm to construct the rest of the timetable taking the delay into account. Again one can choose to use the same approach as in the adaptive search and only alter the timetable where needed to reduce the chance of communication problems.

# 5 Results

In this section the results will be discussed. First the performance of the exact model and the effect of the variable reduction methods is presented. This is followed by the benchmarking of the GRASP. The effectiveness and application of the bottleneck identification method is discussed in section 5.3. Finally section 5.4 contains the results of a case study based on a real-life problem supplied by Ab-Ovo.

## 5.1 Exact model

To test the effect of the variable reduction and bounding methods discussed in sections 4.2 and 4.3, 100 random instances of the time train timetabling problem were generated. For each instance size in table 1, ten instances were generated. These instances consist of a network with the specified number of nodes. The ratio of station to other decision points is kept at around 0.5. Trains are then generated with random departure times origin and destination stations, and departure times somewhere on the planning horizon. Table 1 contains the average computation time needed by CPLEX version 12.9 to reach the optimal solution and a solution with a proven gap of at most 5%. The mean computation times for this can be found in the columns with header "Time opt" and "Time opt 5%. It also contains the computation times after variable reduction and bound improvement were applied. These can be found in the columns with header "Time opt red" and "Time 5% red". These tables are used to asses the effectiveness of the variable reduction techniques. To set the $\gamma, \delta, \eta$ and $\epsilon$ parameters a small sensitivity analysis was performed. Values where selected in such a way that the maximum number of variables were removed, and the bounds set as tightly as possible, without altering the optimal solution to the instances. The results sensitivity analysis can be found in tables 2 and 3. The $\gamma and \delta$ parameters were set to 4, as for this value all of the instances tested had the same objective value and solution as running without the reduction, indicating that the resulting solution is feasible. The instances with 100 trains are omitted as not all could be solved to optimality and thus the check whether the solution was equal for different values of gamma could not always be performed. The relation between feasibility and the selected value of gamma is not inherently dependent on the instance size, so it is safe to assume that the selected value of $\gamma = 4$ will yield feasible solution for most instances. Table 3 contains the same results but for the choice of $\eta$. It contains the number of test instances that kept the same objective after adding the variable restrictions of section 4.2 with different values for eta. The most efficient restriction that did not change the optimal objective value found in any of the test cases is possible at $\eta = 3$ and $\epsilon = 4$. These selected parameter values were used to construct the results of table 1.

**Table 1:** Mean and (standard deviation) of computation times in seconds. $*0.5\% gap$

| **#trains** | **#nodes** | $\gamma\&\delta$ | **Time opt** | **Time opt red** | **Time 5%** | **Time 5%red** |
|---|---|---|---|---|---|---|
| **10** | **10** | 4 | 0.18(0.04) | 0.05(0.02) | 0.16(0.04) | 0.04(0.01) |
| **20** | **10** | 4 | 0.40(0.12) | 0.14(0.03) | 0.25(0.03) | 0.09(0.02) |
| **30** | **10** | 4 | 2.23(0.94) | 0.55(0.14) | 0.87(0.32) | 0.35(0.14) |
| **40** | **20** | 4 | 18.23(5.36) | 2.98(1.11) | 7.29(2.24) | 0.8(0.13) |
| **50** | **20** | 4 | 355(63) | 32(14) | 72(25) | 10(2) |
| **60** | **20** | 4 | 897(266) | 50(12) | 99(32) | 28(5) |
| **70** | **30** | 4 | 1355(344) | 169(32) | 248(41) | 69(19) |
| **80** | **30** | 4 | 3892(873)* | 589(124)* | 928(233) | 188(22) |
| **90** | **30** | 4 | 5229(1282)* | 1606(367)* | 2234(617) | 437(154) |
| **100** | **30** | 4 | - | 3345(754)* | 6243(1343) | 1989(432) |

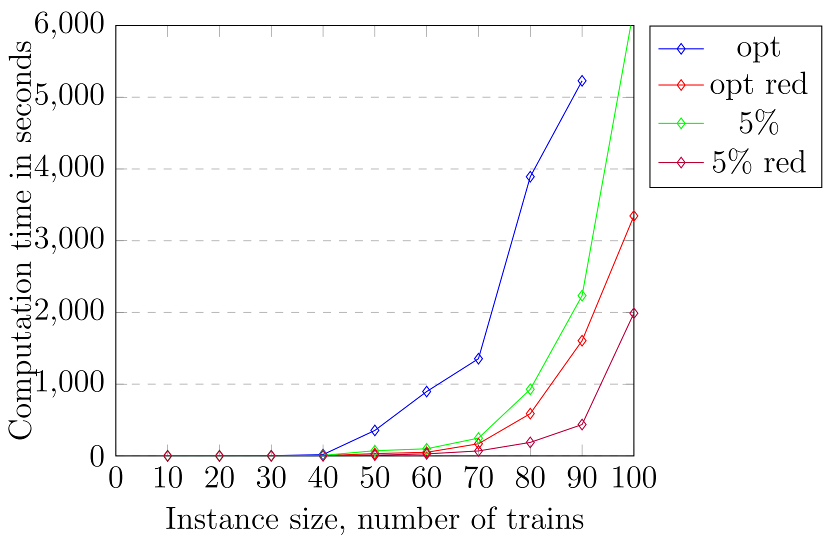

**Figure 5:** Computation time in seconds for different instance sizes

**Table 2:** Number of test instances with feasible optima for different values for $\gamma$

| **#trains** | **#nodes** | $\gamma = 1$ | $\gamma = 2$ | $\gamma = 3$ | $\gamma = 4$ |
|---|---|---|---|---|---|
| **10** | **10** | 0 | 4 | 10 | 10 |
| **20** | **10** | 1 | 6 | 10 | 10 |
| **30** | **10** | 0 | 3 | 8 | 10 |
| **40** | **20** | 1 | 5 | 10 | 10 |
| **50** | **20** | 2 | 5 | 9 | 10 |
| **60** | **20** | 0 | 3 | 7 | 10 |
| **70** | **30** | 1 | 5 | 8 | 10 |
| **80** | **30** | 0 | 4 | 9 | 10 |
| **90** | **30** | 0 | 3 | 8 | 10 |

**Table 3:** Number of test instances that kept the same optimum after variable restriction

| **#trains** | **#nodes** | $\eta = 1, \epsilon = 4$ | $\eta = 2, \epsilon = 4$ | $\eta = 3, \epsilon = 4$ |
|---|---|---|---|---|
| **10** | **10** | 0 | 7 | 10 |
| **20** | **10** | 0 | 9 | 10 |
| **30** | **10** | 0 | 8 | 10 |
| **40** | **20** | 0 | 7 | 10 |
| **50** | **20** | 0 | 8 | 10 |
| **60** | **20** | 0 | 8 | 10 |
| **70** | **30** | 0 | 9 | 10 |
| **80** | **30** | 0 | 7 | 10 |
| **90** | **30** | 0 | 8 | 10 |

Figure 5 contains a plot of computation time against instance size. The applied variable reduction methods bring down the computation time required significantly, solving some instances up to 20 times faster than the regular model. With these reduction techniques, the exact model can efficiently solve instances up to 100 trains within one hour on average.

## 5.2 GRASP

### 5.2.1 Parameter choices

Before discussing the performance of the GRASP some clarification must be given about the parameter values that were used. Regarding the meta-parameters, the performance of the GRASP will be presented for an increasing number of iterations so only the local search and simulated annealing parameters need to be decided on. Regarding the local search, it seems sensible to have the number of iterations depend on the size of the problem instance. More trains imply more potential conflicts, because more pairs of trains may need to compute for use of the same tracks. As the size of our neighbourhood depends heavily on the number of conflicts, the first improvement search method is used, and the number of local search iterations is limited to the number of trains to avoid spending too much time in the adaptive search. This value was selected as after trying multiple values it offered a good balance between finding improvements in a neighbourhood, while not taking up unreasonable amounts of computation time as compared to the greedy heuristic. For the process of simulated annealing, the results presented in this section compare the performance of the GRASP without simulated annealing and with an annealing probability of 0.1. Other values were tried but did not yield any additional results that were of interest. Now to decide on the parameters of the greedy algorithm itself, the decision comes down to whether we prefer local optimality or not. The only parameters that still need to be set are the probability to select track switching as a solution and the probabilities deciding which train has to wait in an overtaking conflict. It is always locally optimal to change tracks if possible, as this does not incur additional waiting time. Likewise letting a slower train wait to let a faster one pass is always locally optimal as not doing so would imply the two would come into conflict again at some later point in the network. After testing the alternatives it appeared best to select the locally optimal path in most cases. The parameter values used in the results shown below are such that if a track switch is possible, it is selected as a resolution method with probability 0.8, and the probability that a fast train has to wait for a slow train is set to 0.1. Multiple other values were tried, and results for parameter values leaning more towards locally less optimal decisions can be found in the appendix. These values proved to be most effective for the instances tested.

### 5.2.2 Performance

Table 4 shows the average percentage gap obtained after a certain number of iterations without using simulated annealing. It also shows the computation time needed to perform a single iteration of the GRASP. The same instances were used as in section 5.1. For smaller instances, the GRASP finds solutions close to the optimal solution quite quickly. As the number of trains increases, the number of iterations needed to obtain solutions with a gap below 10 % also increases. Overall the GRASP outperforms the exact model without variable reduction. It finds solutions of comparable quality much faster. The performance of the GRASP is slightly worse than the exact model. As the instance size increases, the

GRASP shows potential in that it finds solutions that are at most 6% worse than the optimal solution consistently. The increase in computation time also is not as steep as for the exact model as shown in Figure 6. This makes it the better candidate to use for instances too large for the exact model to handle even after variable reduction. For instances up to 100 trains, it seems there is a small preference for the exact model with variable reduction. Figure 7 shows the improvement gradually decreases in the number of iterations and eventually converges towards zero.

**Table 4:** Average percentage gap and computation time per iteration for a different number of iterations

| **Trains/Iterations** | **10** | **20** | **30** | **40** | **50** | **60** | **70** | **80** | **90** | **100** | **Time(s)** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **10** | 5.3 | 2.0 | 0.4 | 0.4 | 0.4 | 0.4 | 0.4 | 0.3 | 0.3 | 0.3 | 0.003 |
| **20** | 8.7 | 2.5 | 1.4 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.020 |
| **30** | 12.3 | 6.5 | 5.3 | 1.2 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.312 |
| **40** | 14.3 | 11.4 | 6.5 | 1.1 | 0.8 | 0.7 | 0.5 | 0.5 | 0.5 | 0.5 | 1.040 |
| **50** | 18.2 | 13.2 | 9.2 | 6.4 | 3.2 | 2.1 | 1.0 | 0.8 | 0.8 | 0.8 | 2.855 |
| **60** | 20.2 | 14.4 | 12.7 | 7.6 | 4.4 | 3.1 | 2.1 | 2.1 | 1.3 | 1.3 | 5.422 |
| **70** | 22.2 | 18.7 | 10.1 | 9.7 | 7.6 | 5.1 | 2.2 | 1.8 | 1.2 | 1.2 | 8.649 |
| **80** | 23.6 | 20.8 | 13.6 | 10.9 | 8.8 | 6.2 | 4.7 | 3.2 | 3.2 | 2.9 | 9.446 |
| **90** | 24.7 | 19.9 | 18.5 | 11.7 | 9.5 | 7.1 | 5.9 | 4.8 | 3.9 | 3.9 | 16.331 |
| **100** | 26.7 | 22.9 | 18.2 | 15.4 | 12.2 | 8.8 | 8.3 | 6.3 | 5.9 | 5.9 | 26.267 |

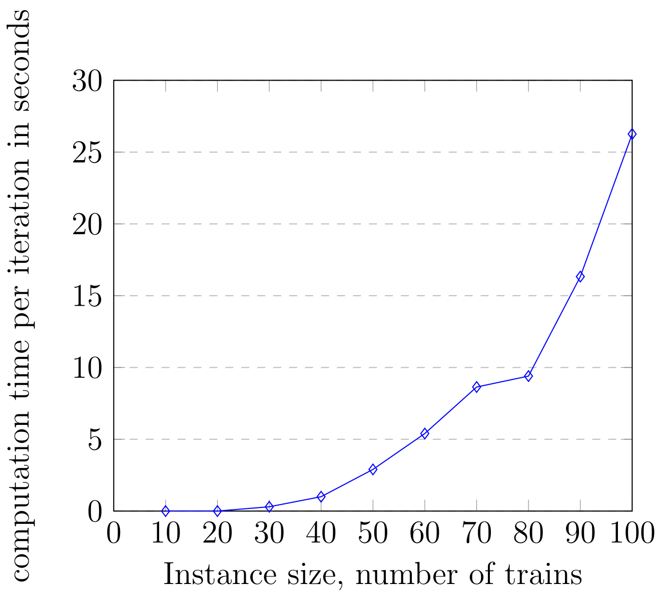

**Figure 6:** Computation time in seconds for different instance sizes

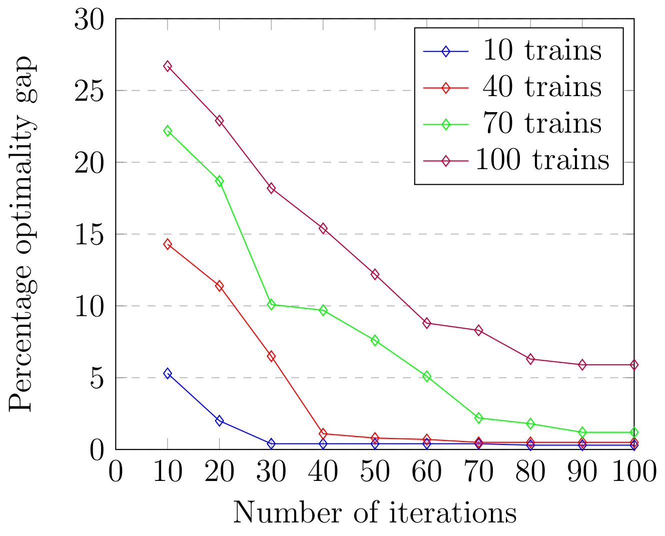

**Figure 7:** Computation time in seconds for different instance sizes

Table 5 contains results for the same instances using simulated annealing. For some instance a slightly better solution is found, but overall the difference is small.

**Table 5:** average percentage gap and computation time per iteration for GRASP results with simulated annealing

| **Trains/Iteraions** | **10** | **20** | **30** | **40** | **50** | **60** | **70** | **80** | **90** | **100** | **Time(s)** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **10** | 5.5 | 2.0 | 0.4 | 0.4 | 0.4 | 0.4 | 0.4 | 0.4 | 0.4 | 0.4 | 0.003 |
| **20** | 9.0 | 2.6 | 1.4 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.022 |
| **30** | 12.2 | 7.8 | 5.4 | 1.2 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.328 |
| **40** | 14.1 | 8.5 | 6.7 | 1.1 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 1.011 |
| **50** | 186 | 13.3 | 8.8 | 6.7 | 3.1 | 2.1 | 0.8 | 0.8 | 0.8 | 0.8 | 2.993 |
| **60** | 19.7 | 16.2 | 10.3 | 7.8 | 4.6 | 2.9 | 1.4 | 1.4 | 1.3 | 1.2 | 5.482 |
| **70** | 21.5 | 17.9 | 10.5 | 9.2 | 7.4 | 5.3 | 2.2 | 1.9 | 1.2 | 1.2 | 8.640 |
| **80** | 22.7 | 21.4 | 14.2 | 11.3 | 8.7 | 6.1 | 4.8 | 3.3 | 3.1 | 2.8 | 9.870 |
| **90** | 25.3 | 20.2 | 19.1 | 11.5 | 9.8 | 6.9 | 5.7 | 4.9 | 3.9 | 3.9 | 14.645 |
| **100** | 27.2 | 22.9 | 19.1 | 15.0 | 12.3 | 8.5 | 8.3 | 6.1 | 6.0 | 5.8 | 24.526 |

The tables containing the results for some other parameter sets leaning more towards not selecting local optimal paths as well as tables containing the standard deviations of the gaps presented in this section can be found in the appendix. To end this section figure 8 shows a visual representation of a timetable constructed by the GRASP for a small instance where four trains were scheduled on half a day from 0:00 A.M. until 11 A.M.

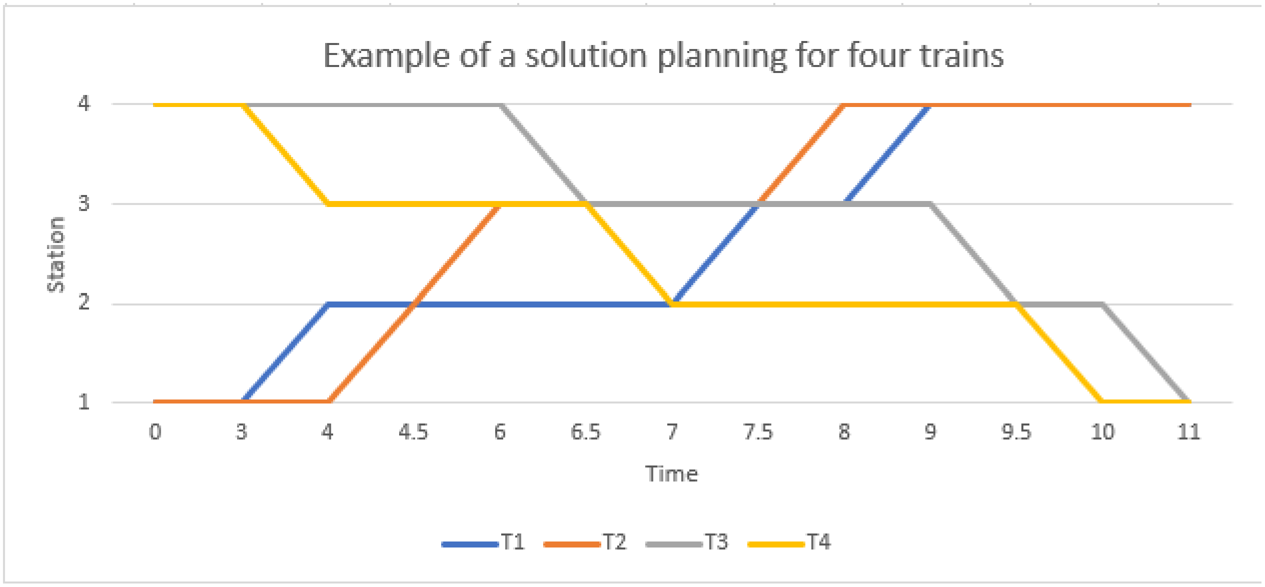

**Figure 8:** Example of a timetable generated by the GRASP

## 5.3 Bottleneck identification

The bottleneck identification method is tested on some small instances. Table 6 contains the results for an instance with 10 decision points and 30 trains. To check whether relieving some pressure from these bottlenecks results in more efficient planning, the smaller instances of section 5.1 and 5.2 are used. Their bottleneck is determined and for the section with the highest bottleneck score an additional piece of track is put in the network. The GRASP is applied again and the average improvement in percentages from adding the piece of track can be found in Table 7. From this we see that adding tracks as identified bottlenecks can improve the objective by up to 8.2%.

**Table 6:** Bottleneck scores B for a small instance after 10 and 50 iterations of the GRASP

| Section | B 10 iterations) | Weighted B10 | B 50 iterations | Weighted B50 |
|---|---|---|---|---|
| **1** | 3.242 | 1.864 | 3.080 | 2.036 |
| **2** | 1.477 | 0.962 | 1.492 | 0.837 |
| **3** | 1.495 | 0.864 | 1.584 | 1.031 |
| **4** | 4.704 | 3.091 | 4.234 | 2.858 |
| **5** | 6.076 | 4.065 | 6.805 | 4.512 |
| **6** | 7.207 | 4.475 | 6.990 | 3.558 |
| **7** | 1.563 | 1.068 | 1.563 | 0.960 |
| **8** | 1.451 | 0.982 | 1.639 | 0.852 |
| **9** | 0.000 | 0.000 | 0.000 | 0.000 |

**Table 7:** Improvement percentage in optimal solution from adding track to bottleneck segment after 10 and 50 iterations of the grasp

| Number of trains | Number of nodes | 10 iterations | 50 iterations |
|---|---|---|---|
| **10** | **10** | 8.233 | 8.233 |
| **20** | **10** | 6.123 | 6.243 |
| **30** | **10** | 5.442 | 5.442 |

## 5.4 Case study: A real-life application

### 5.4.1 Case description

The data for this case study was supplied by Ab Ovo. The goal is to use the GRASP to create a timetable for a week. The network under considerations consists of 64 decision points, 11 of which are switches, 17 are stations and the remaining 36 are passing loops. 227 trains need to be scheduled in total. There are two different speed classes of trains and not all trains are allowed to wait at all passing loops. Most trains have a few required stops on their trajectory. The assumptions made in section 3.2 are all satisfied.

### 5.4.2 Lower bound

The instance specified above is much too large to be handled by the exact formulation of section 4.1. Even after variable reduction the model could not find a feasible solution after 12 hours of running time. To discuss the performance of the GRASP a lower bound on the total travel time is needed. A lower bound can be obtained by taking the expected total travel time of all trains given that they do not encounter any conflicts along the way. This lower bound will most likely be quite a bit lower than any optimal value could ever be. To put this into perspective, the instances of section 5.1 and 5.2 are once again considered. For these instances the average percentage difference between the optimal objective value and this naive lower bound were computed and are presented along with their standard deviation in table 8 below.

**Table 8:** The average percentage difference and its (standard deviation) between the lower bound and optimal solution

| **Trains** | **10** | **20** | **30** | **40** | **50** | **60** | **70** | **80** | **90** |
|---|---|---|---|---|---|---|---|---|---|
| **d** | 12.59(7.46) | 14.13(7.22) | 11.23(6.54) | 14.37(7.88) | 13.95(6.37) | 17.64(8.23) | 14.58(7.53) | 13.28(8.83) | 17.94(8.98) |

There is no clear relation between instance size and the percentage difference between the optimal solution and the proposed lower bound. The total mean over all percentage differences is 14.41%. This can be used to give a slightly more detailed indication of the GRASP performance.

#### 5.4.3 Results

The following tables contain the results of the case study. To avoid results based on luck or misfortune, the algorithm was applied to the instance 10 times and the average results are presented below.

**Table 9:** Instance details

| **trains** | **nodes** | **mean t per iteration(s)** | **LB (hours)** | **Expected LB(hours)** |
|---|---|---|---|---|
| 227 | 64 | 368 | 2933.450 | 3356.160 |

**Table 10:** Average gap and adjusted gap in percentages after a number of iterations

| **Iterations** | **Average best objective(hours)** | **Average gap%** | **Average adjusted gap%** |
|---|---|---|---|
| **10** | 4746.617 | 61.8 | 41.4 |
| **20** | 4556.182 | 55.3 | 35.8 |
| **30** | 4393.878 | 49.8 | 30.9 |
| **40** | 4152.212 | 41.5 | 23.7 |
| **50** | 4051.992 | 38.2 | 20.8 |
| **60** | 3943.924 | 34.45 | 17.5 |
| **70** | 3845.980 | 31.0 | 14.5 |
| **80** | 3746.570 | 27.7 | 11.6 |
| **90** | 3708.328 | 26.4 | 10.5 |
| **100** | 3699.671 | 26.1 | 10.2 |

Thus in 100 iterations our GRASP finds a solution on average with a gap of at most 26% and this gap is more likely to actually be around 10%. On average it takes around 10 hours to obtain a solution of this quality.

# 6 Discussion and conclusion

In this section, the results will be discussed further and the research questions posed in section 3.4 will be answered. This is followed by suggestions for further research and a summarizing conclusion.

## 6.1 Discussion

The constructed MIP formulation of the train timetabling problem can solve small instances efficiently. The results in table 1 show that upwards of 90 trains the instance size becomes too large to handle for the model as is. This exact method can be improved drastically in terms of speed by applying better variable bounds and reducing the number of variables based on the likelihood of trains coming into conflict. This strategy does have one significant drawback. Using the variable reduction may accidentally relax the problem and this result in an infeasible solution. Using a lenience parameter $\gamma = 4$ resulted in no such violations among our 100 generated test cases. However, this not a guarantee that no instances exist or could be purposely constructed for which a higher value for gamma is needed to prevent infeasibilities. A pleasant incidental is that using a low value for gamma, the exact solving method can be used to construct lower bounds for large instances. On the other hand setting tight variable bounds, while potentially improving the speed at which the optimal solution is found, restricts the problem, and may thus exclude the optimal solution from the search space if the value of the corresponding $\eta$ parameter is set too low. Solving a tight restriction of the problem may provide a useful upper bound for the optimal objective value. A potential caveat is that restricting the domain of the variables too much may make the problem infeasible, thus the usefulness of this restriction is limited for finding upper bounds.

While being useful for instances up to 100 trains, the exact model is no longer able to consistently find good solutions for larger instances like the one provided in section 5.4. The MIP formulation was used to determine the effectiveness of the GRASP scheme for instances up to 100 trains and is useful from a benchmarking perspective. While after reduction the exact model outperforms the GRASP for smaller instances, much larger real-life cases can still be solved by the GRASP consistently with decent accuracy as the results obtained in section 5.4 indicate that for this case study the found solution is at most 26% worse than the optimal. The GRASP can solve instances up to 100 trains with an average gap of at most 6% within 90 minutes. For larger instances like the case study of section 5.4 more iterations and computation time may be needed to obtain a satisfactory solution. A GRASP has not been used for the train timetabling problem before in the current literature and this research shows that the scheme has the potential to be a good alternative to existing methods. One of the main advantages of the GRASP scheme is that it can be used to both generate quick solutions by using just a few iterations and good solutions if more time is available. This makes it an excellent candidate to use both for disruption resolution as well as constructing efficient planning beforehand. Besides being a direct solution method, the GRASP could be used in combination with existing meta-heuristics like the one presented in Ghoseiri & Morshedsolouk (2005) to generate large pools of starting solutions. In terms of parameter choice, the results in section 4 indicate that parameters favouring locally optimal decisions provide better solutions. In the context of the train timetabling problem, this does not come as a big surprise as having a faster train stuck behind a slow one for longer than necessary is rarely ever expected to yield good results. The results in table 4 and 5 also indicate that the application of simulated annealing in this procedure did not significantly improve the solution quality. This could be because the neighbourhood used by the adaptive search is already diverse enough. Changing the outcome of a single conflict could radically impact the possible resolutions for future conflicts. It could also cause or prevent conflict from occurring. This results in a large neighbourhood with a complex structure, even without using simulated annealing. The case study shows that the GRASP can be used to solve large real life instances of the problem with decent quality within hours. In about ten hours a gap of at most 26% is found, which is more likely to be around 10%. Given that our exact formulation was unable to find a feasible solution this makes the GRASP the clear winner to be used for large real-life instances. One final advantage of the GRASP is that it can be executed in parallel. Only the current best solution needs to be saved and compared, but every iterations can be performed independently from the others. Thus using enough computing power many iterations could be performed simultaneously.

The solutions obtained by the GRASP can also be used to identify bottlenecks in the network. The weighted bottleneck score often gives the same conclusion as the non-weighted score. This is to be expected as the trajectory for the trains remains the same and bottlenecks are expected to occur at heavily used segments of the network without passing loops or double track. After adding a track to the identified bottleneck section, significant improvements to the objective value could be realised. This procedure's usefulness is limited to network extension decision support for fixed demand, as the bottleneck score depends heavily on the planned trajectories for the train. To use the method for finding general bottlenecks on the network one could average the segment scores over multiple instances. However, for this goal using methods like the one suggested in Jamili (2018) may prove more effective.

## 6.2 Ideas for further research

Both the exact model and the GRASP scheme are currently specified to problem instances where there are at most two tracks on each section of the network. To make these techniques more generally applicable they could be extended to also include more complex network structures. The exact model needs only minimal adjustment as it already uses binary variables to indicate if a train is on a track on a segment, and variables can simply be added for any additional tracks. The GRASP can be adapted to accommodate multiple tracks by changing the track switching solution algorithm so that a train can switch to multiple tracks, but can never switch back to a track is has already moved away from. Section 4.6 provides some explanation on how to adjust the GRASP to be applicable to situations involving random disturbances and delays. It could be interesting to compare the performance of the GRASP in a setting with delays based on historical data against the performance without any delays. An interesting extension of the bottleneck identification method could be to devise a decision rule based on the additional cost of adding an extra track to a segment. An iterative scheme could be designed to construct an efficient planning while identifying the bottleneck segment, on which an extra track is added if the benefit of this would outweigh the cost over a certain operational period. This way a more optimal network could be designed and scenarios could be tested to estimate benefits of extending the network. Finally, an interesting application of the exact model would be to use it in a dynamic programming inspired context. One could split up the trains into different priority classes, and solve the timetabling problem for the highest priority trains first, then setting their timetable as fixed and solving for the next class. This could be a nice way to incorporate priority into the model and could make for a heuristic that could solve large instances quickly if the priority groups are small enough. An alternative way to incorporate priority into the exact model is to add additional weight to the travel time of trains with a higher priority so that when minimizing the total travel time the priority trains will not wait as long as the others unless the difference in objective is significant. Similarly, priority could be included in the GRASP by altering the decision rules for which train waits to avoid a conflict. These rules could be designed so that trains with a higher priority have a lower probability of having to wait.

## 6.3 Conclusion

Efficient timetabling is of incredible importance for cost-effective transportation. In this thesis, a new exact model for the train timetabling problem was formulated that accommodates single and double-track segments, passing loops, capacity limitations, and heavy traffic in both directions. This model is improved upon in terms of efficiency, size and computation time by applying variable reduction and bound restricting methods. This led to significant improvements in performance. This model is able to effectively tackle instances up to 100 trains. Next, a GRASP is designed to solve the same problem. After investigating different parameter sets, parameters favouring locally optimal resolutions of potential conflicts performs best. This means favouring fast trains overtaking slow trains when possible and using available track switching yields better results. The exact model is used to benchmark the performance of the GRASP. For instances up to 100 trains, the GRASP performs better than the original exact model in terms of computation time but is outperformed by the exact model after variable reduction and bound improvements. The difference becomes smaller as instance size becomes larger and for the larger instances solutions could be found by the GRASP in slightly more than one hour with a gap of at most 6%. Simulated annealing was applied to improve the adaptive search but this did not change the results significantly. A bottleneck identification method was incorporated in the GRASP and for small instances it was shown that bottleneck segments could be identified using this method. Extending the network to alleviate some pressure on these bottlenecks led to an improvement in the objective value up to 8%, though it must be said that this value is highly dependent on the instance and the network the method is being applied to. When applied to a large real-world case, the GRASP performed reasonably well, attaining a gap with an upper bound of 26.12% in little over 10 hours. This gap is likely to be much smaller. From this, it can be concluded that the GRASP scheme defined in this research could be used by companies to construct efficient timetables for their networks. The GRASP can also be used to generate quick feasible solutions when unforeseen disruptions to the schedule happen. The improved exact model can be used to solve smaller instances on part of the network. This model explicitly allows for heavy two-way traffic, passing loops, varying train speed and station capacity. This should allow adaptation to real-life cases with relatively limited adjustments. Concluding, the methods developed in this research provide an interesting addition to existing methods, can handle complex network structures and can provide adequate solutions to real-life problems within reasonable time. Let them be used to keep cargo transport on the right track.

# 7 Appendix

**Table 11:** GRASP results with higher probability of letting fast trains wait

| **T\I** | **10** | **20** | **30** | **40** | **50** | **60** | **70** | **80** | **90** | **100** | **time(s)** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **10** | 6.56 | 2.14 | 0.82 | 0.43 | 0.43 | 0.43 | 0.43 | 0.42 | 0.39 | 0.39 | 0.003 |
| **20** | 9.06 | 2.67 | 1.42 | 0.34 | 0.32 | 0.30 | 0.30 | 0.303 | 0.29 | 0.29 | 0.019 |
| **30** | 11.95 | 7.58 | 5.25 | 1.33 | 0.69 | 0.36 | 0.33 | 0.32 | 0.30 | 0.30 | 0.303 |
| **40** | 13.01 | 8.22 | 6.98 | 1.05 | 0.72 | 0.55 | 0.53 | 0.52 | 0.52 | 0.52 | 1.113 |
| **50** | 19.838 | 13.596 | 9.207 | 7.104 | 3.328 | 2.226 | 0.864 | 0.842 | 0.84 | 0.77 | 2.622 |
| **60** | 22.34 | 17.09 | 9.60 | 7.30 | 4.31 | 3.22 | 2.99 | 1.39 | 1.37 | 1.25 | 6.048 |
| **70** | 22.18 | 19.93 | 11.12 | 10.68 | 8.36 | 5.15 | 4.15 | 2.27 | 1.24 | 1.19 | 9.936 |
| **80** | 25.96 | 20.19 | 12.97 | 11.56 | 8.99 | 6.76 | 5.66 | 4.75 | 3.30 | 3.01 | 9.870 |
| **90** | 28.58 | 21.38 | 20.24 | 12.93 | 11.31 | 10.79 | 9.49 | 7.90 | 6.82 | 5.33 | 18.728 |
| **100** | 26.968 | 24.44 | 23.24 | 16.59 | 12.93 | 11.63 | 9.61 | 8.99 | 8.98 | 8.68 | 36.663 |

**Table 12:** GRASP results with higher probability to not use track switching

| **T\I** | **10** | **20** | **30** | **40** | **50** | **60** | **70** | **80** | **90** | **100** | **time(s)** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **10** | 9.88 | 3.51 | 0.72 | 0.72 | 0.72 | 0.69 | 0.66 | 0.64 | 0.63 | 0.56 | 0.004 |
| **20** | 12.23 | 4.25 | 2.11 | 0.56 | 0.51 | 0.50 | 0.47 | 0.47 | 0.43 | 0.41 | 0.020 |
| **30** | 17.69 | 12.89 | 9.43 | 1.73 | 0.56 | 0.51 | 0.490 | 0.48 | 0.48 | 0.41 | 0.36 |
| **40** | 20.63 | 14.1 | 11.77 | 2.02 | 0.85 | 0.83 | 0.79 | 0.77 | 0.71 | 0.68 | 1.102 |
| **50** | 27.48 | 22.18 | 17.39 | 11.26 | 4.48 | 3.34 | 1.45 | 1.39 | 1.26 | 1.22 | 2.793 |
| **60** | 36.54 | 24.49 | 15.38 | 11.43 | 8.01 | 4.68 | 2.26 | 2.12 | 2.07 | 1.84 | 5.400 |
| **70** | 31.80 | 30.33 | 17.54 | 16.00 | 10.49 | 8.87 | 3.38 | 2.99 | 2.25 | 1.65 | 10.368 |
| **80** | 36.77 | 34.40 | 24.57 | 17.39 | 13.17 | 8.43 | 6.86 | 4.93 | 4.73 | 4.45 | 9.870 |
| **90** | 40.10 | 34.22 | 26.7 | 21.88 | 14.49 | 10.15 | 7.96 | 7.15 | 5.62 | 5.34 | 22.84 |
| **100** | 38.47 | 35.958 | 28.67 | 24.84 | 17.32 | 14.69 | 12.01 | 11.53 | 8.53 | 8.24 | 46.66 |

**Table 13:** Standard deviations of the gaps of table 4

| **T\I** | **10** | **20** | **30** | **40** | **50** | **60** | **70** | **80** | **90** | **100** |
|---|---|---|---|---|---|---|---|---|---|---|
| **10** | 1.256 | 0.462 | 0.112 | 0.088 | 0.094 | 0.109 | 0.106 | 0.121 | 0.118 | 0.118 |
| **20** | 2.002 | 0.563 | 0.378 | 0.078 | 0.066 | 0.076 | 0.091 | 0.082 | 0.091 | 0.086 |
| **30** | 2.806 | 2.063 | 1.285 | 0.270 | 0.080 | 0.077 | 0.073 | 0.091 | 0.074 | 0.082 |
| **40** | 3.015 | 2.054 | 1.600 | 0.314 | 0.138 | 0.130 | 0.129 | 0.140 | 0.128 | 0.111 |
| **50** | 5.324 | 3.234 | 2.813 | 1.936 | 0.760 | 0.599 | 0.212 | 0.222 | 0.218 | 0.196 |
| **60** | 6.141 | 4.274 | 2.695 | 1.938 | 1.331 | 0.775 | 0.397 | 0.319 | 0.335 | 0.374 |
| **70** | 6.216 | 4.277 | 3.060 | 2.134 | 2.071 | 1.160 | 0.512 | 0.401 | 0.327 | 0.321 |
| **80** | 6.077 | 4.790 | 3.983 | 3.078 | 2.203 | 1.426 | 1.351 | 0.768 | 0.928 | 0.718 |
| **90** | 6.519 | 5.049 | 4.533 | 2.691 | 2.712 | 1.793 | 1.623 | 1.368 | 0.887 | 1.190 |
| **100** | 7.139 | 6.412 | 4.508 | 3.565 | 3.172 | 2.003 | 2.200 | 1.402 | 1.552 | 1.697 |

**Table 14:** Standard deviation of the gaps of table 8

| **T\I** | **10** | **20** | **30** | **40** | **50** | **60** | **70** | **80** | **90** | **100** |
|---|---|---|---|---|---|---|---|---|---|---|
| **10** | 1.903 | 0.487 | 0.104 | 0.113 | 0.091 | 0.117 | 0.104 | 0.102 | 0.092 | 0.094 |
| **20** | 2.062 | 0.662 | 0.403 | 0.087 | 0.075 | 0.074 | 0.089 | 0.087 | 0.074 | 0.079 |
| **30** | 3.168 | 1.799 | 1.181 | 0.370 | 0.078 | 0.080 | 0.073 | 0.071 | 0.072 | 0.078 |
| **40** | 3.834 | 1.808 | 1.849 | 0.293 | 0.147 | 0.116 | 0.152 | 0.150 | 0.140 | 0.153 |
| **50** | 5.852 | 3.773 | 2.141 | 1.829 | 0.907 | 0.662 | 0.200 | 0.229 | 0.206 | 0.219 |
| **60** | 4.968 | 4.316 | 2.641 | 2.098 | 1.261 | 0.959 | 0.407 | 0.344 | 0.403 | 0.368 |
| **70** | 5.045 | 5.779 | 2.807 | 2.614 | 1.881 | 1.417 | 0.589 | 0.414 | 0.337 | 0.279 |
| **80** | 5.841 | 5.754 | 3.694 | 3.091 | 2.719 | 2.011 | 1.424 | 0.839 | 0.783 | 0.799 |
| **90** | 7.217 | 5.186 | 5.718 | 3.039 | 2.590 | 2.800 | 2.657 | 2.270 | 1.842 | 1.532 |
| **100** | 5.933 | 5.622 | 6.565 | 3.773 | 3.104 | 2.405 | 2.763 | 2.380 | 1.709 | 2.649 |

**Table 15:** Standard deviation of the gaps of table 9

| **T\I** | **10** | **20** | **30** | **40** | **50** | **60** | **70** | **80** | **90** | **100** |
|---|---|---|---|---|---|---|---|---|---|---|
| **10** | 2.372 | 1.184 | 0.185 | 0.195 | 0.150 | 0.236 | 0.167 | 0.241 | 0.208 | 0.228 |
| **20** | 3.303 | 1.392 | 0.592 | 0.140 | 0.133 | 0.094 | 0.144 | 0.140 | 0.145 | 0.135 |
| **30** | 4.511 | 3.720 | 2.712 | 0.583 | 0.109 | 0.162 | 0.124 | 0.162 | 0.133 | 0.139 |
| **40** | 6.294 | 3.951 | 4.151 | 0.668 | 0.226 | 0.209 | 0.195 | 0.251 | 0.180 | 0.211 |
| **50** | 6.802 | 7.207 | 5.783 | 2.732 | 1.165 | 1.119 | 0.278 | 0.365 | 0.344 | 0.411 |
| **60** | 10.962 | 6.122 | 4.077 | 3.449 | 2.823 | 1.568 | 0.792 | 0.742 | 0.574 | 0.425 |
| **70** | 11.053 | 8.084 | 5.570 | 5.322 | 3.278 | 2.285 | 1.070 | 0.744 | 0.626 | 0.462 |
| **80** | 14.620 | 10.063 | 7.310 | 6.173 | 4.248 | 2.150 | 1.887 | 1.255 | 1.234 | 1.615 |
| **90** | 9.423 | 6.950 | 7.615 | 7.603 | 3.805 | 2.767 | 2.509 | 2.110 | 1.530 | 1.883 |
| **100** | 9.529 | 12.888 | 8.942 | 8.459 | 5.370 | 4.025 | 3.893 | 3.315 | 2.494 | 2.636 |

**Table 16:** Standard deviation of the gaps of table 5

| **T\I** | **10** | **20** | **30** | **40** | **50** | **60** | **70** | **80** | **90** | **100** |
|---|---|---|---|---|---|---|---|---|---|---|
| **10** | 1.500 | 0.514 | 0.096 | 0.108 | 0.103 | 0.103 | 0.110 | 0.116 | 0.091 | 0.111 |
| **20** | 2.693 | 0.586 | 0.356 | 0.084 | 0.064 | 0.086 | 0.074 | 0.092 | 0.080 | 0.083 |
| **30** | 3.050 | 2.126 | 1.446 | 0.288 | 0.091 | 0.063 | 0.091 | 0.068 | 0.085 | 0.078 |
| **40** | 3.236 | 2.330 | 1.914 | 0.258 | 0.119 | 0.145 | 0.122 | 0.132 | 0.118 | 0.111 |
| **50** | 5.616 | 3.433 | 2.076 | 1.714 | 0.768 | 0.473 | 0.198 | 0.210 | 0.233 | 0.229 |
| **60** | 4.923 | 4.528 | 2.778 | 1.860 | 1.384 | 0.677 | 0.362 | 0.332 | 0.287 | 0.308 |
| **70** | 6.236 | 3.929 | 2.521 | 2.281 | 2.160 | 1.366 | 0.656 | 0.454 | 0.336 | 0.285 |
| **80** | 5.494 | 6.106 | 3.863 | 3.374 | 1.941 | 1.413 | 1.464 | 0.753 | 0.974 | 0.689 |
| **90** | 6.335 | 5.958 | 5.717 | 3.468 | 2.962 | 1.687 | 1.487 | 1.104 | 1.121 | 1.170 |
| **100** | 7.966 | 5.840 | 5.071 | 4.097 | 3.142 | 2.367 | 2.034 | 1.543 | 1.465 | 1.704 |

# 8 References

Albrecht, A. R., Panton, D. M., & Lee, D. H. (2013). Rescheduling rail networks with maintenance disruptions using problem space search. *Computers & Operations Research*, *40*(3), 703–712.

Cacchiani, V., Caprara, A., & Toth, P. (2008). A column generation approach to train timetabling on a corridor. *4OR*, *6*(2), 125–142.

Cai, X., Goh, C., & Mees, A. I. (1998). Greedy heuristics for rapid scheduling of trains on a single track. *IIE transactions*, *30*(5), 481–493.

Caprara, A., Monaci, M., Toth, P., & Guida, P. L. (2006). A lagrangian heuristic algorithm for a real-world train timetabling problem. *Discrete applied mathematics*, *154*(5), 738–753.

Castillo, E., Gallego, I., Ureña, J. M., & Coronado, J. M. (2011). Timetabling optimization of a mixed double-and single-tracked railway network. *Applied Mathematical Modelling*, *35*(2), 859–878.

Feo, T. A., & Resende, M. G. (1995). Greedy randomized adaptive search procedures. *Journal of global optimization*, *6*(2), 109–133.

Ghoseiri, K., & Morshedsolouk, F. (2005). An ant colony system heuristic for train scheduling problem.

Ghoseiri, K., Szidarovszky, F., & Asgharpour, M. J. (2004). A multi-objective train scheduling model and solution. *Transportation research part B: Methodological*, *38*(10), 927–952.

Harbering, J., Ranade, A., Schmidt, M., & Sinnen, O. (2015). Single track train scheduling. *Planning a Public Transportation System with a View Towards Passengers' Convenience*, 151.

Higgins, A., Kozan, E., & Ferreira, L. (1997). Heuristic techniques for single line train scheduling. *Journal of heuristics*, *3*(1), 43–62.

Jamili, A. (2018). Computation of practical capacity in single-track railway lines based on computing the minimum buffer times. *Journal of Rail Transport Planning & Management*, *8*(2), 91–102.

Landex, A. (2008). *Methods to estimate railway capacity and passenger delays.* Technical University of Denmark.

Liebchen, C., & Stiller, S. (2009). Delay resistant timetabling. *Public transport*, *1*(1), 55–72.

Marques-Silva, J. P., & Sakallah, K. A. (1999). Grasp: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, *48*(5), 506–521.

Silva, J. P. M., & Sakallah, K. A. (2003). Grasp—a new search algorithm for satisfiability. In *The best of iccad* (pp. 73–89). Springer.

Vianna, D. S., & Arroyo, J. E. C. (2004). A grasp algorithm for the multi-objective knapsack problem. In *Xxiv international conference of the chilean computer science society* (pp. 69–75).

Wahlborg, M. (2004). Banverket experience of capacity calculations according to the uic capacity leaflet. *WIT Transactions on The Built Environment*, *74*.

Yang, Y., Huang, P., Peng, Q., Jie, L., & Wen, C. (2019). Statistical delay distribution analysis on high-speed railway trains. *Journal of Modern Transportation*, *27*(3), 188–197.