Utrecht University

Balancing Costs and Driver Satisfaction in Cargo Train Driver Scheduling

Mats Gottenbos

Master's Thesis

Supervisors:
dr. J.A. Hoogeveen
dr. ir. J.M. van den Akker

Department of Information and Computing Sciences
Utrecht University
Netherlands
July 2022

Abstract

This study considers the problem of scheduling cargo train drivers to perform a given set of activities, which is an important problem for our client Rail Force One. In this multi-objective problem, a trade-off must be found between monetary costs and driver satisfaction. Solutions must adhere to many real-world constraints, like maximum shift lengths, minimum resting times, contract hours, and driver qualifications. They must also deal with multiple types of drivers with different starting locations, varying salary rates throughout the time frame, and differing scheduling rules.

To solve this problem, we developed a simulated annealing algorithm to find high-quality schedules. This algorithm includes a simplified but high-performance robustness model that focuses on the idle time between consecutive activities, as well as a satisfaction score comprised of twelve criteria for each driver. By using varying weights between estimated costs and satisfaction throughout the simulated annealing search, a Pareto-optimal front of schedules is created representing the trade-off between the two objectives.

Experimental analysis using real-world data shows that the results of the algorithm are feasible and produced in acceptable runtimes. The generated schedules outperform the client's human-made schedules in cost-effectiveness, driver satisfaction and robustness. Consequently, the algorithm can reduce planning staff workload while increasing the client's profits and their drivers' satisfaction.

Preface

This thesis is submitted in partial fulfilment of the requirements for the degree of Master of Science in the Department of Information and Computing Sciences in the Graduate School of Natural Sciences of Utrecht University.

This study is a research project performed for the company Rail Force One, where I was employed as an intern for the duration of the thesis.

I want to express great gratitude to my supervisors at the university, Han Hoogeveen and Marjan van den Akker, for all the invaluable feedback and advice they extended to me during this thesis. Furthermore, I want to thank the helpful people at Rail Force One for their assistance. In particular, I sincerely thank my supervisor at the company, Sander Vermeijs, for his tireless efforts to provide me with all the information and data needed to complete this project. Finally, I wish to convey a great deal of appreciation to everyone else that advised and supported me throughout this thesis.

Contents

1

Introduction

This study is conducted for the company *Rail Force One*[1], part of *Rail Innovators Group*, a cargo train operator active in many European countries. An essential part of the company's operations is the planning of train routes, vehicles and drivers. This thesis focuses on the problem of finding an optimal assignment of cargo train drivers to activities. These activities include train driving, but also related tasks like locomotive inspections. The solution must optimise monetary costs, robustness and driver satisfaction, while also adhering to a series of real-world constraints.

The driver assignment must be determined for around 400 activities each week. An important factor is that the drivers live in different locations, meaning they must be treated in a heterogeneous manner. Since most travel time is compensated, minimising travel leads to major financial advantages. This task is complicated by driver preferences, such as the desire to drive a varied set of routes. Ensuring driver satisfaction is important to the company as it allows them to better retain and attract driving staff, leading to long-term growth.

In the current situation, the driver assignment is manually created by driver planning staff. These planners have just two days to create each week's schedule, since this is the time between the finalising of the orders and a contractually mandated planning deadline. Because the task is so complex, the planning staff regularly needs to work considerable overtime in order to finish it. In addition, despite their expertise, a human planner would likely produce a sub-optimal schedule, given the size of the search space and the limited time available. This study is aimed at developing an algorithm that gives better solutions in a shorter amount of time.

The remainder of this thesis is organised in the following way. In Chapter 2, we give a detailed description of the problem. In Chapter 3, we review the previous literature that relates to this problem. Chapter 4 describes the algorithm used to find schedules optimised for monetary costs. Chapter 5 incorporates robustness into the algorithm and Chapter 6 extends it to include several driver satisfaction criteria. Next, in Chapter 7, we present the results of computational experiments performed using the algorithm. Finally, in Chapter 8, we give the conclusion of the paper and propose directions for future research.

[1] https://www.railforce.one/

2

Problem description

We denote the problem examined in this paper as the Cargo Train Driver Scheduling Problem (CTDSP). In the CTDSP, an assignment must be found of cargo train drivers to activities with the highest possible quality. Three criteria should be taken into account when determining the quality of an assignment: its monetary costs, its driver satisfaction, and its robustness. All three criteria will be detailed further in this section.

A list of activities is given, including their starting and ending locations and their planned start and end times. As its solution, the system should provide multiple schedules, based on different weightings of the monetary cost and driver satisfaction criteria. Each schedule must also have a decent level of robustness. A complete schedule consists of many individual driver schedules that combine to cover all activities. These driver schedules are comprised of shifts, which are sets of activities performed consecutively as a working day.

It is important to note that the schedules determined by the algorithm are not expected to be used without adjustments. Instead, the planning staff will always be involved in the process. They are responsible for selecting the best schedule out of the different cost-satisfaction trade-offs provided by the algorithm. They then use this schedule as a useful foundation for a final schedule, making modifications where needed.

2.1 Constraints

Being a practical problem, the CTDSP is bound by many real-world constraints. This section describes these constraints, whose definitions are based on the expert knowledge of the client's staff.

2.1.1 Qualifications

Train driving activities may require particular qualifications of the driver performing it. Only qualified drivers can be assigned to such an activity. The main type of qualification considered are route qualifications, which are necessary to drive on each corresponding route.

2.1.2 Internal and external drivers

There are two types of drivers: internal and external. There is a fixed number of internal drivers. These are meant to perform the vast majority of activities, since they are the cheapest for the company.

There are several companies offering external drivers for hire. Per company, these drivers may be split into separate groups that each have their own set of qualifications.

Each group has a minimum and a maximum number of shifts that the client must hire from them each week. External drivers are more flexible but also more expensive than internal drivers, meaning they should be used to perform activities that would be inconvenient for internal drivers. Only separate shifts need to be scheduled for external drivers, not full weekly schedules. There is no need to optimise the satisfaction of external drivers.

2.1.3 Travel times

At the start of their shift, drivers use their personal car to drive to the starting location of their first activity. Internal drivers do so from their home address, while external driver companies instead have a set starting address for all of their drivers. If drivers perform activities in multiple locations in the shift, they use a company-owned pool car to drive between stations as needed. At the end of the shift, the drivers first drive back to the starting location of their first activity in the pool car, after which they drive to their home or starting address using their personal car.

Alternatively, it is possible to schedule a stay at a hotel between two shifts, which means the driver does not need to drive to their starting address between those shifts. When a driver stays in a hotel, they travel by pool car from the last activity of the previous shift to the hotel and later from the hotel to the first activity of the next shift. They pick up their personal car at the end of the next shift. The travel time for a hotel stay is considered to be the travel time between the last station before and the first station after the stay, plus half an hour for the detour via the hotel. We assume this travel time is evenly split between the shifts before and after the hotel stay. A hotel stay is only possible between shifts on consecutive days, meaning the time between the shifts cannot be more than 24 hours.

All travel in pool cars is considered part of the shift, while travel in personal cars is considered travel time outside of the shift. Note that the vehicle scheduling of pool cars is handled outside of this model.

2.1.4 Driver availability

Drivers may be unavailable during parts of the week because of urgent business, like hospital appointments. They cannot be assigned any activities during these parts of the week. Note that these unavailabilities are different from non-urgent requests for time off, which are handled under the *driver satisfaction* criterion in Section 2.2.2.

2.1.5 Driver shift counts and contract hours

Internal drivers have a maximum number of five shifts per week. Exceeding this number of not allowed. They also have a fixed number of contract hours per week, which may differ between drivers. These contract hours concern all shift lengths are defined in Section 2.2.1. It is not required to adhere to the contract hours perfectly, but the planned number of hours should generally be at most 20% higher or lower than the number of contract hours.

If there is a situation where the solution could be significantly improved by slightly exceeding the contract hour bounds, then the system should still give such a solution as an option. For the shift counts, no such exceptions can be made.

2.1.6 Safety regulations

A driver's schedule must adhere to both labour laws and company safety policies. The regulations are as follows:

1. Driver shifts may not be longer than 10 hours excluding personal car travel times, nor longer than 12 hours including personal car travel times.
2. Night shifts, which are by law shifts with an hour or more worked time between 0:00 and 6:00, have a maximum duration of 8 hours excluding travel times and 10 hours including travel times.
3. The resting period between a driver's consecutive shifts must be at least 11 hours long. If a shift ends after 2:00, the resting period must be at least 14 hours long.

Note that there is no need to schedule breaks, since opportunities for breaks are already included in the activity durations as defined by the client.

2.2 Objectives

2.2.1 Minimising monetary costs

It is an important objective to minimise the monetary costs of the schedule. Note that minimising the costs leads to maximum profit since the revenue is constant. This is because the revenue of each activity is unaffected by considerations like its selected driver or departure time, and is therefore unchanged by any changes the model can make.

The monetary costs of the model consist of the costs per driver. The components of these costs are detailed next.

Salary rates

The major component in the driver costs are the salaries. In reality, hourly wages differ between individual drivers and external driver companies, but the model will work with estimated averages per group of drivers. This is done to avoid situations where the lowest-paid drivers are almost always assigned to the longest activities, which is undesirable since these drivers are also the least experienced. The wages only differ in the following ways:

1. International drivers have a higher hourly wage than national drivers. A driver is considered international if they have route qualifications in multiple countries.
2. External drivers have a higher hourly wage than internal drivers.
3. Each driver group has a higher hourly wage during certain parts of the week, such as at night and during weekends.
4. Each driver group has a fixed hourly wage for travel time, which is the same for the entire week.

Shifts

The length of a shift is computed as the time between the start of the first activity and the end of the last activity, plus any time spent travelling in pool cars outside of this period. As such, this length is composed of activity time, waiting time between activities, travel time between activities, travel time to pick up a personal car, and travel to and from hotels. Drivers are paid the relevant salary for this shift length. Internal drivers are paid for a minimum of six worked hours per shift, even if the actual shift is shorter. For external drivers, this minimum shift length is eight hours. Additionally, there is an estimated cost for the company for each kilometre driven using pool cars, representing fuel and other costs.

The travel time of a shift, meaning driving from and to the driver's starting location in their personal car, is paid at the fixed travel rate. For internal drivers, all but the first hour of travel time per shift is compensated. For external drivers, the full travel time is compensated.

Additionally, there is an estimated cost incurred for each scheduled hotel stay.

2.2.2 Maximising driver satisfaction

The system should aim to maximise the drivers' satisfaction levels. The following objectives should be included with regard to a driver's satisfaction level:

1. Assign activities with the desired level of route variation. Drivers may request high variation, low variation, or may not have a preference.
2. Minimise the travel time by personal car before and after shifts.
3. Minimise the difference between the worked time and the contract time of the driver.
4. Assign shifts of the desired length. Drivers may prefer long or short activities.
5. Maximise robustness in the driver's schedule.
6. Minimise the number of night shifts.
7. Minimise the number of weekend shifts.
8. Minimise the number of hotel stays.
9. Respect requests for specific (parts of) days off. Note that urgent unavailabilities are instead handled as hard constraints, as described in Section 2.1.4.
10. Distribute the number of consecutive shifts without free days evenly, aiming for sequences of five consecutive shifts.
11. Assign two consecutive free days instead of separate single days.
12. Maximise the resting time between shifts.

Each driver can have different priorities among the satisfaction criteria, which the algorithm should take into account. The total satisfaction of each driver is considered of equal importance.

The objectives should take into account data from past weeks, not just the week currently being planned. While achieving perfect satisfaction will be impossible, especially

in conjunction with the other objectives, the system should aim to distribute undesirable circumstances between drivers. Therefore, it should not just look at the average of the drivers' satisfaction levels, but also at their minimum or a similar measure. It should particularly be avoided that the same drivers are assigned unsatisfactory schedules for multiple weeks in a row.

2.2.3 Maximising robustness

The system should also attempt to maximise the robustness of the schedule when minimising monetary costs. Robustness indicates the ability of a schedule to handle operational disruptions. The goal is to make a schedule where individual disturbances, like a train being delayed or a driver being suddenly unable to come into work, have as little effect as possible on other parts of the schedule. For example, it is positive for the robustness to have a larger waiting time between consecutive activities performed by the same driver, as this waiting time can act as a buffer when disruptions occur.

The measure of robustness should be impacted more when disturbances spread between multiple projects than when they do between activities of the same project.

Some information is available to aid the handling of disruptions. Firstly, an estimated risk of delays is given for each activity. Secondly, it is known that some drivers are more flexible regarding delayed trains than others. As such, it is desirable to assign flexible drivers to activities with higher risks of delay. This can be taken into account when estimating the robustness level of a schedule.

3

Literature review

This study is a continuation of a line of research for Rail Force One started by van Strien (2021). Whereas van Strien studied the assignment of locomotives to a given train schedule, we examine the assignment of drivers to these locomotives.

This literature review focuses on three different areas. First, we explore the most fundamental part of the problem, which is about creating a driver schedule with minimum monetary cost. Second, we review literature concerning the inclusion of driver satisfaction in the scheduling process, as this is the second criterion of the problem. Finally, we discuss previous work on scheduling with robustness in mind, which is the third criterion.

3.1 Scheduling

The CTDSP has an extensive set of constraints. At its core, however, it bears resemblance to several well-known problems. The most fitting of these are the Bus Driver Scheduling Problem, the Multiple-Depot Vehicle Scheduling Problem and the Airline Crew Scheduling Problem. These three problems each have similarities and differences to the CTDSP. The methods used to solve them could potentially be combined to inform a solution method for the CTDSP. We will discuss previous studies on each problem, focusing on the application of more general techniques like linear programming and local search, instead of custom algorithms and heuristics. This is done since the former techniques are likely easier to adapt to other problems such as the CTDSP.

3.1.1 Bus Driver Scheduling Problem

The Bus Driver Scheduling Problem (BDSP) is about assigning drivers to cover a trip schedule. The problem is most commonly applied to bus drivers, as its name suggests, but it can be used equally well with trains and other forms of public transportation. In the BDSP, the trip schedule is given, meaning that the start and end times of all trips are known. The objective is usually to minimise the sum of shift costs, the number of required drivers, or a combination of the two.

In the various literature, many different solution approaches to the BDSP have been explored. Linear programming with column generation has been applied by Fores (1996) to bus driving and by Alfieri et al. (2007) to train train driving. Desaulniers et al. (2002) explored several strategies to speed up the column generation method. Kwan and Kwan (2007) introduced a hybrid method of column generation and an iterative heuristic to speed up the method for large instances.

Local search techniques have also been explored. Tabu search was applied to bus driving by Lourenço et al. (2001) and Shen (2001), as well as to train driving by Laplagne

(2008). More recently, Hanafi and Kozan (2014) have applied simulated annealing to train driving, whereas Peng et al. (2015) have used multi-objective simulated annealing for bus driving.

The BDSP has clear similarities to the CTDSP. Like the CTDSP, the BDSP focuses on assigning drivers, meaning that studies tend to incorporate labour safety constraints. Most studies apply a maximum shift length for drivers, similar to the CTDSP. Studies also often include meal breaks, but these are not relevant for the CTDSP. The study by Laplagne also included some constraints to ensure the contract hours of drivers can be met.

However, there are also some clear differences between the BDSP and the CTDSP. The most important is the fact that the BDSP uses homogeneous drivers, meaning there are no differences in starting or ending location between drivers, nor in their contract hours, qualifications or availability. The requirement to take these factors into account fundamentally changes the algorithms required to solve the problem, since it necessitates the tracking of individual drivers rather than creating universal options for shifts.

3.1.2 Multiple-Depot Vehicle Scheduling Problem

The Multiple-Depot Vehicle Scheduling Problem (MDVSP) is a problem of covering trips with vehicles, instead of with drivers. In this case, too, the trip schedule is known. There are two variants of this problem, the heterogeneous and the homogeneous variant, each with a different constraint regarding the depots. In the heterogeneous variant, vehicles must end at the day at their starting depot. In the homogeneous variant, it is only required that each depot ends the day with the same number of vehicles it started with, but these do not have to be the same individual vehicles. For the CTDSP, the heterogeneous variant is more relevant. Given these constraints, an assignment must be found such that each trip is covered by a vehicle. The objective is usually to minimise the sum of shift costs, the number of required vehicles, or a combination of the two.

The MDVSP has also been approached in many different ways. Column generation was examined by, among others, Ribeiro and Soumis (1994), Hadjar et al. (2006), Kulkarni et al. (2018) and Domínguez-Martín et al. (2018). In terms of local search algorithms, Lim and Zhu (2006) applied simulated annealing and Renaud et al. (1996) applied tabu search. Moreover, Pepin et al. (2009) performed a study comparing a truncated branch-and-cut algorithm, a Lagrangian heuristic, a truncated column generation method, a large neighbourhood search heuristic using column generation for neighbourhood evaluation, and a tabu search heuristic. They concluded that column generation was the best method when enough computation time is available, while large neighbourhood search was the best option when there is a need for good solutions in relatively fast computation times.

The MDVSP has key elements that are similar to the CTDSP. Most importantly, the non-homogeneous drivers in the CTDSP can be treated similarly to the non-homogeneous vehicles in the MDVSP. The fact that drivers live at different locations can be viewed as each internal driver having its home as its depot, while external drivers would have the company offices as their depots.

On the other hand, the MDVSP lacks some components of the CTDSP that the BDSP does have. Since the MDVSP schedules vehicles instead of drivers, it does not include constraints about shift lengths, resting period lengths, or contract hours. A close variant of the Vehicle Scheduling Problem, however, the Electric Vehicle Scheduling Problem (e-VSP), may be of use in this respect. In the e-VSP, vehicles have batteries with a capacity and a required charging time. The capacity could be used to model the maximum shift length in the CTDSP, while the charging time could be used to enforce the minimum

resting time between shifts. Important to note is that when applied to the CTDSP, no mid-shift charging is possible.

To solve the e-VSP, van Kooten Niekerk et al. (2017) have developed a column generation approach, while Wen et al. (2016) offer an approach with large neighbourhood search. The multi-depot version of the e-VSP has been studied less extensively, but Wang et al. (2021) provide an approach that combines a genetic algorithm with column generation.

3.1.3 Airline Crew Scheduling Problem

The Airline Crew Scheduling Problem (ACSP) is about covering a given flight schedule with sufficient numbers of crew members. The problem generally includes multiple labour constraints, though implementations differ between studies. Most studies work with a maximum shift length and a minimum resting time. Some studies also include multiple home bases, where groups of crew members must start and end their schedule.

The ACSP, too, has seen many different techniques used in literature. A column generation approach was pursued by Gustafsson (1999) and subsequently improved by Borndörfer et al. (2006) and Bayer (2012), among others. Simulated annealing has been applied by Emden-Weinert and Proksch (1999) and Lučic and Teodorovic (1999), but it appears no more studies have used this approach since.

Several important constraints of the CTDSP are present in the ACSP, such as maximum shift lengths and minimum resting times. Some studies also incorporate different home bases, which are relatively similar to the different starting and ending locations of drivers in the ACSP.

On the other hand, there remain key differences between the ACSP and CTDSP. Most importantly, the ACSP does not include constraints on contract hours. Furthermore, the ACSP requires multiple crew members to be assigned per flight, whereas the CTDSP only requires a single driver per train.

3.2 Driver satisfaction

Studies that look at crew satisfaction in scheduling are not very popular in literature, but recent years have seen a rise in interest in the subject. Studies usually look at satisfying crew preferences, minimising schedule unfairness, or both. Preferences cover the desires of drivers to be assigned to certain activities over others, while fairness is about balancing workload or unpopular activities between drivers.

In the area of train driver scheduling, both Jütte et al. (2017) and Porokka et al. (2017) focus on fairness. They aim to balance the number of unpopular activities between drivers. The former study does this using a manually determined minimum and maximum number of unpopular shifts per driver, whereas the latter includes unfairness in the objective function.

Studies about crew preferences have focused on the application of airline crew scheduling. Moudani et al. (2001) create an algorithm where crew members can select preferred or non-preferred flights, where a minimum value of preference satisfaction is enforced as a constraint. A study by Zhou et al. (2020) uses preferences on both preferred flights and off-periods, as well as fairness in workload distribution. It creates a bi-objective model to optimise for these two criteria, but ignores monetary costs completely. Quesnel et al. (2020) also look at preferred flights and off-periods, as well as fairness. It includes the preferences in the objective function alongside other costs. They note that constraints are included to enforce fairness, but they do not detail these constraints.

In the CTDSP, crew preferences and fairness are important. However, another important factor in the crew satisfaction is activity variation. The seemingly only study to take this subject into account is from Meijer et al. (2017). As a measure of variation, they use the number of unique kilometres assigned to each depot.

3.3 Robustness

Previous literature provides many different ways of measuring and maximising robustness. Most studies focus on machine scheduling problems, where the objective is usually to minimise the makespan of the schedule. In recent years, there have also been a number of studies on measuring robustness specifically in real-world timetabling, where the specifics of these problems are incorporated more extensively. We discuss both types of measures.

3.3.1 Robustness in machine scheduling

Most robustness measures in machine scheduling centre around the concept of *slack*. The *total slack* of an activity, as introduced by Jorge Leon et al. (1994), is the maximum amount of time that this activity can be delayed by, where delaying other activities is allowed, as long as the makespan of the schedule does not change. Similarly, Al-Fawzan and Haouari (2005) define an activity's *free slack* as the amount of time it can be delayed without delaying any other activities. For both types of slack, a schedule's robustness can be estimated by taking either the average or the sum of the slack of all activities.

Other studies introduce different slack-based estimations. Chtourou and Haouari (2008) introduce several alternatives to using the sum of free slacks. They propose weighting each activity's slack by its number of successors, replacing the free slack with a binary function, or adding an upper bound on an activity's slack based on the activity duration. Kobylański and Kuchta (2007) introduce taking the minimum over the activities' free slacks, as opposed to the sum, when working with deadlines. Khemakhem and Chtourou (2013) show experimentally that a good estimation for a schedule's robustness is its *slack sufficiency*, which compares the free slack of each activity to a fraction of the duration of the activity and its predecessors.

Wilson et al. (2014) introduce two so-called *flexibility* measures for schedules with deadlines. The first one is *naive flexibility*, defined as the sum over all activities of the difference between the activity's earliest and latest possible starting times. This is similar to the measure of total slack, but computed with respect to the deadline, rather than the makespan. The second measure they define is *concurrent flexibility*. This uses the notion of interval schedules, which specify an interval for each activity such that it can freely start at any time in this interval, independent of other activities and respecting the deadline. The concurrent flexibility of a schedule is the maximal sum of interval lengths from any possible interval schedule.

Continuing with this approach, van den Broek et al. (2018) propose to maximise the minimum interval length instead of the sum, to ensure a more even distribution of the interval lengths. Furthermore, they introduce a new measure called the *minimum weighted path slack*. This measure takes the minimum over all paths in the activities' partial ordering, of the path's slack divided by its number of activities. They also introduce a related measure called *minimum probability of completion* that takes the minimum over each path in the partial ordering, of the path's probability of being completed before the deadline. This probability is approximated by assuming each activity's processing time is normally distributed and summing over the distributions of the path's activities.

Lastly, there exist measures based on approximations of the schedule's makespan. Passage et al. (2016) propose an efficient version, where the activities are evaluated in topological order and each makespan distribution up to an activity is calculated as the maximum over its predecessor's distributions. The measure is defined as the probability that the makespan remains within the deadline.

3.3.2 Robustness in real-world timetabling

Multiple methods for measuring robustness in real-world timetables have been developed. There are several high-level techniques, which are implemented in detail for particular problems.

The first one is *stochastic programming*, which was applied to airline crew scheduling by Yen and Birge (2006). Stochastic programming is very accurate and flexible, but it is often too computationally intensive for real-world instances. Additionally, it requires information about the probability distributions of activity durations, which may be hard to gather or estimate.

A simplified version of stochastic programming, designed by Diepen et al. (2012), still uses probabilistic information about delays but is much faster computationally. They look exclusively at the idle time between each pair of consecutive activities, using a scoring function to assign larger penalties to shorter periods of idle time. The authors showed that this technique, when used for assigning flights to airport gates, gives high-quality results in a short computation time.

An alternative approach is *robust optimisation*, which was applied to airline crew scheduling by Antunes et al. (2019). Robust optimisation is much faster computationally and does not require probability distributions, but its results are often too conservative for practical use. For this reason, Fischetti and Monaci (2009) introduce a variant called *light robustness*, which optimises robustness within a given bound for the objective value, making it more flexible. Light robustness was later generalised by Schöbel (2014).

A newer approach is *recoverable robustness*, introduced by Liebchen et al. (2009). This method takes into account specified possibilities for recovery after disruptions. It is meant to be more flexible than robust optimisation, while being less computationally intensive than stochastic programming.

4

Scheduling

The examined literature shows a multitude of possible methods to solve problems like the CTDSP. Ideally, we would determine the single most effective approach to solve the CTDSP. However, studies that compare different methods on the same relevant problem are rare. The only relevant comparative study is one by Pepin et al. (2009), performed on the MDVSP. That study concludes that column generation leads to the highest quality solutions, but large neighbourhood search may be used to find decent solutions faster. Still, it remains unknown to what extent these results may be generalised to different problems. Additionally, the study does not include some other interesting methods like simulated annealing.

This shows that, for a problem without previous research, selecting an optimal method with certainty is not possible. Even so, the large number of relevant studies that use column generation or one of the various local search approaches suggest that these are generally viable options for a problem of this kind.

The relative performance of local search algorithms like simulated annealing, large neighbourhood search and tabu search seems to be similarly hard to predict. As such, this choice will most likely come down to personal preference and expertise.

Since there is no definitive method with the best performance, it makes sense to factor in other aspects important to this study. Since the problem has a rather large number of constraints, it makes sense to use a method where this many constraints can easily be implemented. Additionally, since the details of the constraints are prone to repeated change during the consultancy phase, a method flexible to implementing such changes is advantageous. Given these considerations, we have implemented a simulated annealing algorithm to solve the CTDSP.

4.1 Simulated annealing

Simulated annealing (SA) is a technique of repeatedly examining randomly chosen changes to a solution. The better the change in cost is, the higher the chance of the change being accepted. By performing millions of iterations in this way, the algorithm is likely to find a high-quality solution.

If the algorithm would accept changes if and only if they decreased the solution cost, it would likely get stuck in a local optimum that is not the global optimum. For this reason, changes that increase the solution cost still have a chance of being accepted. This chance is determined by the cost difference and a key variable called the *temperature*. The temperature starts high and slowly decreases, meaning the algorithm starts off volatile and stabilises over time. The higher the temperature, the more likely a cost increase is

accepted. For a cost difference of Δ and temperature of t, the acceptance probability is defined by the following formula:

$$P(\Delta) = \begin{cases} 1 & \text{if } \Delta \leq 0 \\ e^{-\frac{\Delta}{t}} & \text{otherwise} \end{cases}$$

Algorithm 1 shows the structure of the SA technique in pseudocode. The following constants are given:

- A cost function $f(\omega)$ for any solution ω
- A neighbourhood function $N(\omega)$ for any solution ω
- A total number of iterations I
- A number of iterations per temperature update J
- An initial temperature t_0
- A temperature update factor $a \in [0, 1\rangle$

Algorithm 1 Simulated annealing

```
1: Select an initial solution ω
2: Set temperature t ← t_0
3: Set the iteration number i ← 0
4: while i < I do
5:     Randomly select a neighbouring solution ω' ∈ N(ω)
6:     Determine cost difference Δ ← f(ω') − f(ω)
7:     if Δ ≤ 0 then
8:         Accept new solution: ω ← ω'
9:     else
10:        With a probability of e^(−Δ/t), accept new solution: ω ← ω'
11:    end if
12:    Update iteration counter: i ← i + 1
13:    Every J iterations, update temperature: t ← a · t
14: end while
```

The algorithm developed in this study uses an initial temperature of $t_0 = 2000$. The temperature is updated every $J = 100,000$ iterations by a factor of $a = 0.97$. The total number of iterations I is determined after an experimental analysis, as described in Section 7.1.

4.2 Initial assignment

Our SA algorithm is initialised with a mostly random assignment. Using the list of activities ordered by starting time, the following is done for each activity:

1. Determine a random order of all internal drivers. Assign the activity to the first internal driver in this order that would not have an overlap violation after doing so.
2. Determine a random order of all external drivers. Assign the activity to the first external driver in this order that would not have an overlap violation after doing so.

3. Given that an overlap violation is inevitable for this activity, assign it to a random external driver of a random type.

The definition of an overlap violation is given in Section 4.4. Although the initial assignment is likely to violate many constraints, this is acceptable due to the system of penalty costs also defined in Section 4.4.

4.3 Neighbourhood operations

A fundamental part of the SA algorithm is the selection of neighbouring solutions. This selection is performed by making small changes to the given solution in a randomised fashion. To do so, SA algorithms often define a set of neighbourhood operations that each perform a specific type of change. The neighbouring solution is determined by selecting a weighted random operation. The algorithm used in this study uses the operations listed below, with their respective selection probabilities. These probabilities were determined through experimentation.

- Assign internal driver (70%). Replace the driver of a random activity with a randomly selected internal driver that is not the current driver.
- Assign external driver (10%). Replace the driver of a random activity with a randomly selected external driver of a randomly selected type, such that the selected driver is not the current driver.
- Swap drivers (19%). Swap drivers of two random activities that do not have the same driver.
- Toggle hotel stay (1%). Select a random activity and add a hotel stay after this activity if there was none, or remove the hotel stay if there was one.

Note that the random selections of the operations are performed without prioritising certain trips or drivers based on the current schedule or the constraint violations that the operation would cause. Constraint violations are instead discouraged through the system of penalty costs discussed in Section 4.4.

When assigning or swapping activities, the new schedules of the affected drivers are automatically recomputed into shifts. Any two consecutive activities are considered to be in the same shift if the waiting time between them is at most six hours.

4.4 Penalty costs

To allow for more flexibility in avoiding local optima, an SA algorithm generally allows intermediate solutions to be invalid. Instead of outright rejecting the invalid solutions, they are instead assigned penalty costs. The more invalid a solution is, the higher the penalty costs should be. The algorithm then adds these penalty costs to the normal costs, using these added values to determine cost differences. In this way, the algorithm allows for invalid solutions, but still pushes the algorithm toward valid ones. This technique has the added advantage of allowing initial solutions to be found trivially. This study uses the following types of penalty costs:

- Overlap. This is violated when a driver is assigned two activities that either overlap directly, or it is impossible to travel between quickly enough. A penalty of 10,000 is given for each violation.

- Shift length. This is violated when a driver's shift exceeds either the maximum shift length excluding travel time or including travel time. A penalty of 1000 is given for each violation, increased by 1000 per minute of violation.
- Resting time. This is violated when the resting time between consecutive shifts of a driver is less than the minimum rest time. A penalty of 1000 is given for each violation, increased by 1000 per minute of violation.
- Internal driver shift count. This is violated when a driver is assigned more shifts than the maximum number of shifts. A penalty of 2000 is given per number of shifts in excess.
- External driver type shift count. This is violated when an external driver type is assigned a number of shifts above or below its allowed range. A penalty of 4000 is given per number of shifts outside the range.
- Invalid hotel stay. This is violated when a hotel stay is assigned after an activity that is either not the end of its shift, or is the end of the schedule. A penalty of 4000 is given for each violation.
- Qualification. This is violated when a driver is assigned to an activity they are not qualified for. A penalty of 4000 is given for each violation.

4.5 Cycles

When simulated annealing is executed for a long time, the temperature eventually becomes so low that the algorithm remains in the current local optimum and no new solutions are found. To allow for continued exploration in such cases, it is a common practice to work with *cycles* in the algorithm that act as partial resets. In this study, we let new cycles start when the temperature has decreased to a threshold set to 0.1. When this happens, the temperature is reset to a randomly selected value between 500 and 3000. As such, some cycles end with a larger partial reset than others. Note that the temperature after a partial reset may be higher than the initial temperature of 2000, which can be useful to force the algorithm out of the local optimum it is in at the end of a cycle.

Additionally, we introduce a 20% chance of performing a hard reset at the end of each cycle. This hard reset fully restarts the algorithm to prevent situations where it might return to similar local optima even after several partial resets.

4.6 Multithreading

To increase the speed of the algorithm, we implement a system of multithreading. Hypothetically, we could approach this by having multiple parallel threads continually read from and write to the same assignment. The high frequency of iterations, however, would mean that threads spend a large share of time waiting on each other's operations. Therefore, this study instead executes an independent instance of the algorithm on each parallel thread. This means the algorithm is essentially executed multiple times in parallel. Because of the highly probabilistic nature of simulated annealing, this approach works very well and allows the algorithm to find high-quality results much faster than it would on a single thread.

5

Robustness

The algorithm described so far bases its optimisation on specified activity and car travel durations. In practice, however, these durations are just estimates, with the actual durations often deviating considerably. Delays in many problems, such as those concerning public transportation, often have lengths in the order of minutes. In cargo train driving, on the other hand, delays are regularly measured in hours.

Ignoring these delays would lead to a schedule that is great in theory, but is almost certain to break down quickly. Delays in the duration of one activity would have a cascading effect on the next activities of that driver, as well as other activities depending on it via precedence relations. This can lead to increased operational costs, as well as missed contractual obligations to clients.

As such, it is crucial for optimising the real-world monetary costs to incorporate robustness optimisation in the algorithm. Robustness is defined as the ability of a schedule to retain its objective value in the face of disturbances.

5.1 Dataset probability distribution

To determine probability distributions that adequately approximate the delays of activities, we analysed historical activity data. Data from one full year was used, during which time 21883 activities were logged. Note that most of these activities are not unique, as tasks are often repeated with a certain frequency.

We define the delay of an activity as its actual duration minus its planned duration. As such, delays can both be positive, meaning that the activity took longer than planned, and negative, meaning it was performed faster.

Figure 5.1 presents the distribution of all activity delays in the data. It shows that the majority of activities have no delay at all. Since we cannot reliably depend on activities to have negative delays, we deem it undesirable for the maximisation of robustness to include these in the computation. Therefore, a negative delay is considered equal to no delay. This leaves 6008 activities with a positive delay, or 27.5% of all activities. Figure 5.2 shows a histogram of specifically these positive delays.

The distribution of positive delays can be approximated with a gamma distribution. To diminish the influence of outliers, delays of more than ten hours are removed from the data used in the fit. This will not result in significant differences for the algorithm, since such large delays have very low probabilities and are therefore not cost-efficient to change the schedule for, if that is even possible.

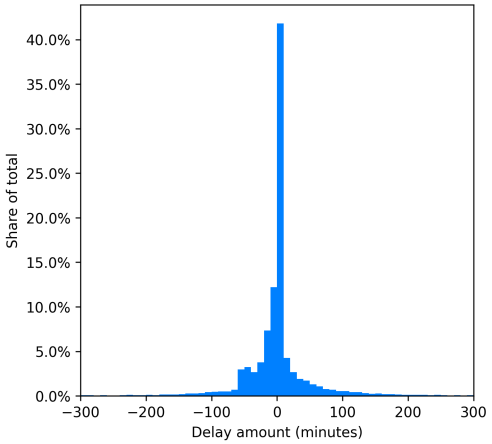


Figure 5.1: Histogram showing the distribution of all delays.

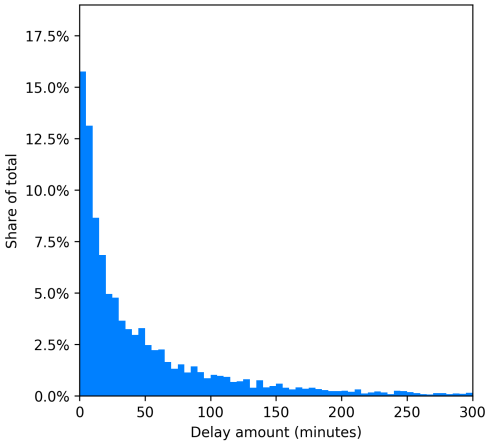


Figure 5.2: Histogram showing the distribution of positive delays.

Fitting a gamma distribution distribution to the data using the *scipy.stats.gamma.fit*[1] function in Python yields parameters $\alpha = 0.6920$ and $\beta = 0.0134$. Figure 5.3 shows this distribution fitted to the histogram of Figure 5.2. Figure 5.4 shows a probability plot comparing the sample data with the distribution. The R^2 value of this distribution is 0.987, meaning it explains 98.7% of the variance in the data. Therefore, the distribution fits the data very well.

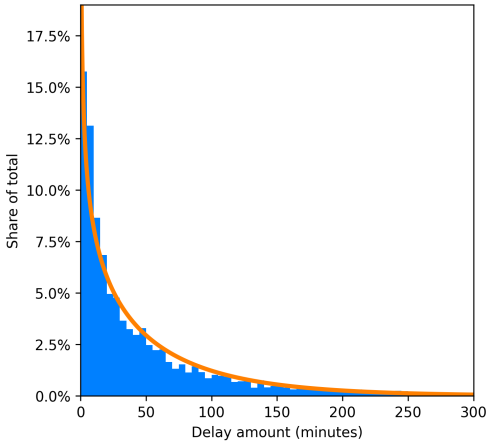


Figure 5.3: The fitted gamma distribution (orange) compared to the histogram of the positive delay distribution (blue).

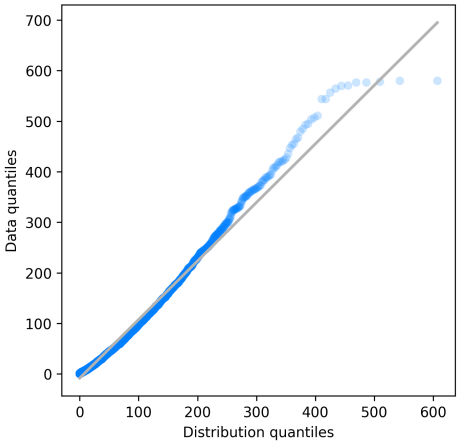


Figure 5.4: Probability plot comparing the fitted gamma distribution (X-axis) to the positive delay data (Y-axis). The 45-degree line is shown in grey. The fit has an R^2 value of 0.987.

5.2 Differences by planned duration

While we now have a distribution for the delays in the entire dataset, not all activities are created equal. A key difference is in the planned duration of the activities. On average, longer activities have longer delays, but the relation is not linear. Figure 5.5 shows a scatter

[1] https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.gamma.html

plot of delay amounts by duration. The amount of data is large enough to approximate this relation with a quadratic function. Using the *numpy.polyfit*[2] package yields Equation 5.1.

$$\mu = -\frac{p^2}{5571} + 0.123p + 37.38 \tag{5.1}$$

Here, p is the planned duration and μ is the mean delay, both in minutes. This curve is included in the plot of Figure 5.5.

Examining the relation between planned duration and the standard deviation of the delay is more difficult. Figure 5.6 shows the standard deviation of the activities within each 30-minute interval of planned durations. There does not seem a clear relation between these values. Therefore, it seems reasonable to assume the standard deviation to be constant, irrespective of the planned duration.

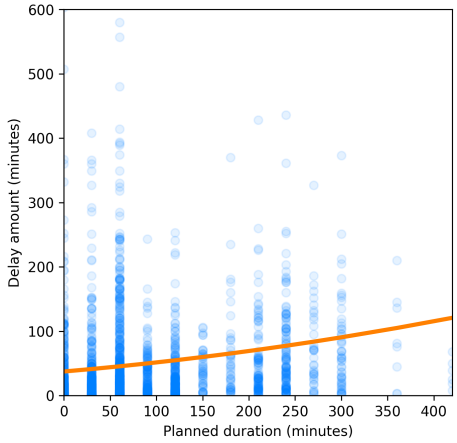


Figure 5.5: Scatter plot of delay amounts by planned duration (blue). A quadratic function is shown in orange.

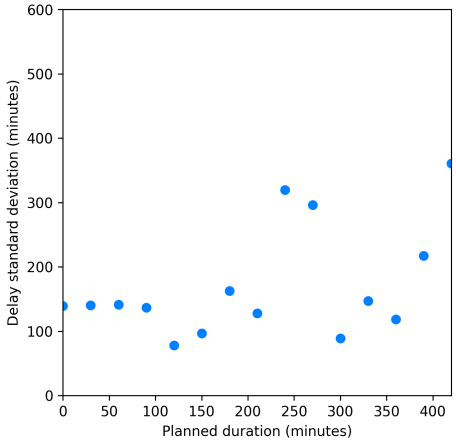


Figure 5.6: Scatter plot of standard deviation of delay amounts. Standard deviation is determined per 30-minute interval of activities' planned durations.

Gamma-distributed data with parameters (α, β) has a mean of $\mu = \frac{\alpha}{\beta}$ and a standard deviation of $\sigma = \frac{\sqrt{\alpha}}{\beta}$. The standard deviation shall be kept constant to that of the probability distribution from Section 5.1. Its value is $\sigma = \frac{\sqrt{0.6920}}{0.0134} \approx 62.3$. This leads to the system of equations 5.2 for a given mean delay μ. This system can be resolved to express α and β in μ, as shown in Equations 5.3 and 5.4. Combined with Equation 5.1, this expresses α and β in the planned duration of an activity p.

$$\begin{cases} \frac{\alpha}{\beta} & = \mu \\ \frac{\sqrt{\alpha}}{\beta} & = 62.3 \end{cases} \tag{5.2}$$

$$\frac{\alpha}{\mu} = \frac{\sqrt{\alpha}}{62.3} \Rightarrow \alpha = \frac{\mu^2}{3879} \tag{5.3}$$

$$\beta = \frac{\alpha}{\mu} = \frac{\mu}{3879} \tag{5.4}$$

[2] https://numpy.org/doc/stable/reference/generated/numpy.polyfit.html

5.3 Differences by activity type

Another important way that activities differ is by their type of task. The clearest differences are visible when comparing train-driving activities with other activities such as inspections and shunting. Figure 5.7 shows large differences in the distribution of all delays. Train-driving activities are delayed 39.0% of the time, whereas non-train-driving activities are delayed in only 16.4% of cases. When looking at just the positive delays in Figure 5.8, however, the distributions are very similar. We can conclude from this that while driving activities have a much higher chance of delay, the delay amounts themselves are all approximated well by the gamma distribution from Section 5.2.

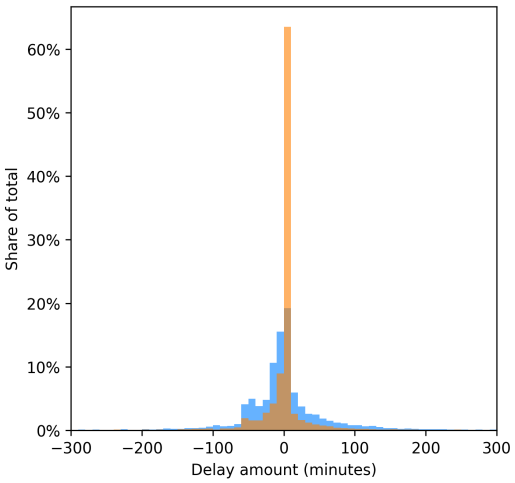


Figure 5.7: Distribution of all delays when split between train-driving activities (blue) and non-train-driving activities (orange).

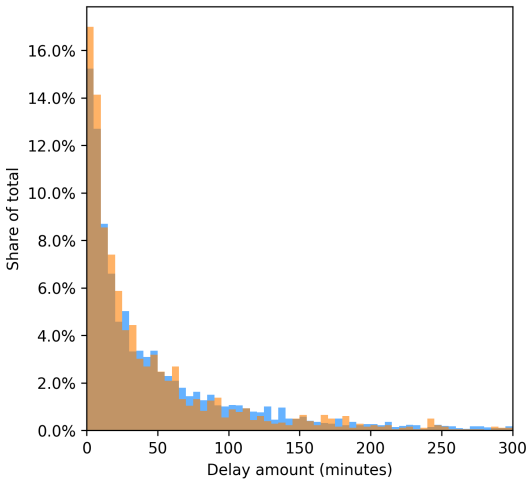


Figure 5.8: Distribution of positive delays when split between train-driving activities (blue) and non-train-driving activities (orange).

5.4 Robustness score method

In order to optimise for robustness, we must define a robustness score that can be determined for any given schedule. As described in Section 3.3, there are many general methods of determining robustness in a schedule. In the CTDSP, a moderately large amount of historical data on planned and actual start and end times of activities is available. As such, reasonably accurate estimations of the probability distributions of delays are possible. This makes it more logical to use stochastic programming than robust optimisation for this problem. However, the high computation time of stochastic programming is very much an issue in this case. When performing many millions of simulated annealing iterations, recursively executing expensive operations on probability distributions would slow the algorithm down too heavily. Therefore, it makes sense to use a simplified version of stochastic programming, similar to that of Diepen et al. (2012).

Diepen et al. defined a robustness measure purely based on the amount of idle time between each consecutive activity. Each period of idle time produced a robustness value using an arctangent-based function, with the total robustness score being the sum of these values. This score was then minimised to obtain a robust schedule.

Directly applying this method to the CTDSP would give two main issues. Firstly,

using an identical function for each period of idle time only works well when all activities are relatively homogeneous in terms of duration and delay risk. In the CTDSP, however, activities range from short inspections to hour-long train driving activities. Data analysis shows that, as one may expect, longer activities tend to have longer delays than shorter activities. In addition, even activities of similar duration can be divided into categories of higher or lower risk based on historical data regarding activities in the same location or on the same route.

Secondly, the method by Diepen et al. optimises a schedule only for robustness, but the CTDSP must optimise for both monetary costs and robustness at the same time. We will create a variant of the method that solves both of these issues.

5.5 Robustness cost definition

We define the robustness cost of a schedule in the following way. For each pair of consecutive activities in a driver's schedule, the appropriate distribution of delays is used to determine the *conflict probability*. There is a conflict if the first activity is delayed by such an amount that this driver can no longer start the second activity on time. This is the case when the delay is larger than the waiting time t_w between the activities, while taking travel time into account. We first determine the *conflict probability if delayed* P'_c using the cumulative gamma distribution of the first activity, as shown in Equation 5.5. This probability is then multiplied by the delay probability P_d of the first activity, as shown in Equation 5.6. The delay probability, as defined in Section 5.3, is 39.0% for train-driving activities and 16.4% for all other activities.

$$P'_c = P\Big(\text{Gamma}(\alpha, \beta) > t_w\Big) \tag{5.5}$$

$$P_c = P_d * P'_c \tag{5.6}$$

This conflict probability is then multiplied with the *expected conflict cost*. This conflict cost depends on the relation between the two activities. In the dataset, activities are grouped into *duties* and *projects*. Duties are sets of consecutive activities that fit well together to form a shift or part of a shift. These duties are predefined by the planning staff and would, for example, combine maintenance checks of a train with the successive driving of the same train. Projects, on the other hand, are defined based on contracts with customers and the resulting financial incentives.

For activities in the same predefined duty, we have no expected conflict cost, because practice has shown that it is very beneficial to robustness to have these related activities be performed by the same driver. Activities of different duties always have an expected conflict cost. If they belong to the same predefined project, this cost is 500, else it is 1000. The product of the conflict probability and the expected conflict cost gives the robustness cost for this pairing of activities.

The sum of all individual robustness costs gives the robustness cost of the full schedule. This robustness cost is the expected additional monetary cost incurred from delays. Therefore, adding the theoretical cost and the robustness cost together gives the total expected cost of the schedule in practice. This total expected cost is used when determining cost differences in the simulated annealing algorithm.

5.6 Robustness cost examples

To illustrate the robustness cost calculations, we give three examples.

5.6.1 First example

Activity pair 1	
09:00 - 10:00	Wagon technical inspection
10:00 - 10:30	Drive train (same duty)

The pair of activities in this example gets no robustness cost, since they are part of the same predefined duty.

5.6.2 Second example

Activity pair 2	
09:00 - 9:30	Drive train
10:30 - 11:30	Drive train (same project)

In this example, the first activity has a planned duration of 30 minutes. Equation 5.1 gives that, if delayed, this activity has a mean delay of:

$$\mu = -\frac{30^2}{5571} + 0.123 \cdot 30 + 37.38 \approx 40.91$$

Using Equations 5.3 and 5.4, we see that this activity has the positive delay amounts of this activity are distributed using a gamma distribution with the following parameters:

$$\alpha = \frac{40.91^2}{3879} \approx 0.4314$$
$$\beta = \frac{40.91}{3879} \approx 0.0105$$

The waiting time between the activities is 60 minutes. Equation 5.5 yields a conflict probability if delayed of 22.4%:

$$P_c' = P\Big(\text{Gamma}(0.4314, 0.0105) > 60\Big) \approx 0.223$$

Next, using the delay probability of 39.0% for train-driving activities, Equation 5.6 gives a conflict probability of 8.7%:

$$P_c = 0.390 \cdot 0.223 \approx 0.0870$$

The expected conflict cost is 500, since these activities belong to the same project. The robustness cost is therefore $0.0870 \cdot 500 = 43.5$.

5.6.3 Third example

Activity pair 3	
09:00 - 11:00	Drive train
11:30 - 12:30	Drive train (different project)

In this example, the first activity has a planned duration of 120 minutes. Equation 5.1 gives that, if delayed, this activity has a mean delay of:

$$\mu = -\frac{120^2}{5571} + 0.123 \cdot 120 + 37.38 \approx 54.72$$

Using Equations 5.3 and 5.4, we see that this activity has the positive delay amounts of this activity are distributed using a gamma distribution with the following parameters:

$$\alpha = \frac{54.72^2}{3879} \approx 0.7719$$
$$\beta = \frac{54.72}{3879} \approx 0.0141$$

The waiting time between the activities is 30 minutes. Equation 5.5 yields a conflict probability if delayed of 53.3%:

$$P_c' = P\Big(\text{Gamma}(0.7719, 0.0141) > 30\Big) \approx 0.533$$

Next, using the delay probability of 39.0% for train-driving activities, Equation 5.6 gives a conflict probability of 20.8%:

$$P_c = 0.390 \cdot 0.533 \approx 0.208$$

The expected conflict cost is 1000, since these activities belong to different projects. The robustness cost is therefore $0.208 \cdot 1000 = 208$.

6

Driver satisfaction

6.1 Satisfaction score

The driver satisfaction value of a schedule is determined by the satisfaction values of each individual driver, which are in turn determined by the performance of that driver's schedule on a set of satisfaction criteria.

We denote by $c_{r,d} \in [0,1]$ the *criterion score* of criterion $r \in \{1, \cdots, R\}$ for driver $d \in \{1, \cdots, D\}$ in a given schedule. The definition of this score depends on the particular criterion and is discussed in Section 6.2.

We define the *driver satisfaction score* $s_d \in [0,1]$ for driver d as the average between the *weighted average driver satisfaction* $\bar{s}_d$ and the *weighted minimum driver satisfaction* $\underline{s}_d$. This ensures that drivers cannot have a very poor satisfaction in some criteria as long as they are compensated by other criteria. It gives the following formula:

$$s_d = \frac{\bar{s}_d + \underline{s}_d}{2}$$

The weighted average driver satisfaction $\bar{s}_d$ is a weighted average of the driver's criterion scores. It uses weights $w_{r,d}$ chosen per criteria r and per driver that sum to 1 for each driver:

$$\bar{s}_d = \sum_r w_{r,d} c_{r,d}$$

The weighted minimum driver satisfaction $\underline{s}_d$ is a minimum over the criteria scores, but with the impact of each criterion scaled by its weight. This is achieved by scaling the amount that each criterion score differs from 50% by the criterion's weight compared to the driver's maximum weight:

$$\underline{s}_d = \min_r \left(0.5 + \frac{w_{r,d}}{W_d} (c_{r,d} - 0.5) \right)$$

Here, the maximum weight of a driver d is denoted as $W_d = \max_r w_{r,d}$. This calculation means that, for example, when a driver with a maximum criterion weight of 20% has a criterion with a score of 30% and a weight of 10%, this criterion counts as 40% in the weighted minimum calculation.

Next, we define the *schedule satisfaction score* $s \in [0,1]$ of a schedule to be the average between the minimum and average over all driver satisfaction scores s_d:

$$s = \frac{\min_d s_d + \sum_d s_d}{2}$$

Combining the minimum and average driver satisfaction in this way ensures that when maximising the schedule satisfaction score, the algorithm is incentivised to increase all drivers' satisfaction scores, but also prevents this from being done at the expense of a few unlucky drivers.

6.2 Criterion scores

In any given timetable, each driver is scored on a set of criteria to determine their satisfaction. The individual criteria are discussed in this section. Each criterion is based on a particular relevant value that is determined from the schedule. For example, *route variation* is based on the number of duplicate routes of that driver in the schedule. Most criteria have thresholds of this value for 0% and 100% satisfaction, with the value and the satisfaction having a linear relation between those thresholds. For ease of notation, we define the following function that performs this linear relation:

$$\lambda(x, \tau_0, \tau_1) = \max\left\{0, \min\left\{\frac{x - \tau_0}{\tau_1 - \tau_0}, 1\right\}\right\}$$

Here, x is the key value for the criterion, τ_0 is the 0% threshold and τ_1 is the 100% threshold.

6.2.1 Individual driver differences

Crucially, drivers' preferences are not uniform. For example, some drivers might want to avoid night shifts, while others may in fact prefer them due to the higher pay rate of those shifts. Still others may not have a preference either way between day or night shifts. The importance of a particular criterion to each driver may also differ. To express these differences, the criteria weights vector $\{w_{1,d}, w_{2,d}, \cdots, w_{R,d}\}$ can differ between individual drivers, with each weight between 0 and 1, as long as each driver's vector sums to 1.

6.2.2 Past satisfaction

While the schedule is determined on a week-by-week basis, a driver's satisfaction on many criteria transcends this timeframe. According to the company's information, drivers are often understanding if they have an unfavourable schedule for one week, but not if they do so for multiple weeks in a row. It is therefore important to include these past scores in the satisfaction score calculation.

For most criteria, this is done by taking what is essentially a rolling average. It is calculated as a weighted average of the criterion scores of this week and the past several weeks, where the most recent weeks are weighted more heavily.

$$c_{r,d} = r_0 * c_{r,d}^{(0)} + r_1 * c_{r,d}^{(1)} + \cdots + r_n * c_{r,d}^{(n)}$$

Here, $c_{r,d}^{(0)}$ is the score of the current week and each $c_{r,d}^{(i)}$ is the past score from i weeks ago. $\{r_0, r_1, \cdots, r_n\}$ is the weight vector for the rolling average with $\sum_{i=0}^{n} r_i = 1$. n is the number of past weeks included in the rolling average.

For some criteria, using a rolling average is not sensible. The formula for those criteria is defined individually in the relevant section.

6.3 Criterion definitions

This section defines each satisfaction criterion used in the algorithm. The best and worst thresholds listed are estimated based on information from the client and experimentation. Note that in practice, the algorithm manages to keep most criterion scores above 50%. This means that while worst-case thresholds may sometimes have rather extreme values, the values in a computed schedule are rarely close to these thresholds.

6.3.1 Shift details

Route variation

Drivers may prefer a high or low level of route variation. The key value x_1 here is the number of times any route is 'repeated' within the time frame. Any two activities that involve train driving and have the same origin and destination are counted as driving the same route. This value is equal to the number of driving activities minus the unique number of routes driven.

For drivers that prefer high variation, the 100% threshold is that there are no duplicate routes, while the 0% threshold is set to ten duplicate routes. The criterion score is linear between these thresholds:

$$c^{(0)}_{1,d,+} = \lambda(x_1, 10, 0)$$

For drivers that prefer low variation, the thresholds are reversed:

$$c^{(0)}_{1,d,-} = \lambda(x_1, 0, 10)$$

This criterion does not use the default rolling average, because routes in past weeks should be directly included in the variation score. Instead, the duplicate routes are counted for a period of the current week and the three weeks before.

Travel time

Drivers may prefer that their travel time x_2 is minimised. Here, we specifically refer to travel time before and after a shift. The travel time between two activities within a shift is therefore excluded.

The best-case threshold is that there is no travel time, while the worst-case threshold is set to 40 hours of travel. The latter is equal to four hours before and after each of the five shifts. The criterion score is linear between these thresholds:

$$c^{(0)}_{2,d} = \lambda(x_2, 40, 0)$$

Contract time accuracy

Drivers have a particular number of working hours per week specified in their contract. While it is usually not possible to match these hours perfectly, drivers often prefer their worked hours to not deviate too much from their contract hours. The worked hours here are the sum of a driver's shift lengths, which exclude the travel times before and after the shift. The key value x_3 is defined as the fraction that the worked hours are either higher or lower than the contract hours. For example, working 44 hours with a 40-hour contract gives $x_3 = 0.1$.

The best-case threshold is no deviation, while the worst-case threshold is set to a deviation of 40%. The criterion score is linear between these thresholds:

$$c_{3,d}^{(0)} = \lambda(x_3, 0.4, 0)$$

Shift lengths

Dividing 40 work hours per week into a maximum of five shifts can be done in multiple ways. For example, it can be achieved both with five shifts of eight hours and with four shifts of ten hours. Drivers may prefer shorter or longer shifts. The key value x_4 is defined as the sum of all shift lengths in excess of eight hours.

For drivers preferring shorter shifts, the best-case threshold is 0 excess hours and the worst-case threshold is 10:

$$c_{4,d,-}^{(0)} = \lambda(x_4, 10, 0)$$

For drivers preferring longer shifts, the thresholds are reversed:

$$c_{4,d,+}^{(0)} = \lambda(x_4, 0, 10)$$

Robustness

Drivers may prefer that the robustness of their schedule is maximised. We measure this robustness using the robustness cost as defined in Section 5.5. We set a best-case robustness cost threshold of 0 and a worst-case threshold of 800.

$$c_{5,d}^{(0)} = \lambda(x_6, 800, 0)$$

6.3.2 Undesirable shifts

Night shifts

Drivers may prefer a low or high number x_5 of night shifts. Whether a shift is a night shift, is defined as a boolean property. Per company rules, a shift is considered a night shift if the majority of its duration is scheduled between 23:00 and 6:00 on a night between two weekdays. Note that this definition is different from the legal definition used in the safety regulations.

For drivers preferring few night shifts, the best-case threshold is no night shifts, while the worst-case threshold is five night shifts. The criterion score is linear between these thresholds:

$$c_{6,d,-}^{(0)} = \lambda(x_6, 5, 0)$$

For drivers preferring many night shifts, the thresholds are reversed:

$$c_{6,d,+}^{(0)} = \lambda(x_6, 0, 5)$$

Weekend shifts

Drivers may prefer a low or high number x_6 of weekend shifts. Whether a shift is a weekend shift, is defined as a boolean property. Per company rules, a shift is considered a weekend shift if the majority of its work time is scheduled between Friday 18:00 and Monday 6:00.

For drivers preferring few weekend shifts, the best-case threshold is no weekend shifts, while the worst-case threshold is three weekend shifts. The threshold of three rather than two is chosen to avoid overly large jumps in the satisfaction, because those would make the system prioritise this criterion more than intended. The criterion score is linear between these thresholds:

$$c^{(0)}_{7,d,-} = \lambda(x_7, 3, 0)$$

For drivers preferring many weekend shifts, the thresholds are reversed:

$$c^{(0)}_{7,d,+} = \lambda(x_7, 0, 3)$$

Hotel stays

Drivers may prefer a low or high number of hotel stays between shifts

For drivers preferring few hotel stays, the best-case threshold is no hotel stays, while the worst-case threshold is four hotel stays. The criterion score is linear between these thresholds:

$$c^{(0)}_{8,d,-} = \lambda(x_8, 4, 0)$$

For drivers preferring many hotel stays, the thresholds are reversed:

$$c^{(0)}_{8,d,+} = \lambda(x_8, 0, 4)$$

Time-off requests

Drivers may file requests for specific free days or parts of days. When they do so, they naturally prefer for such requests to be accepted. The key value $x_9 \in [0, 1]$ is the fraction of requests that are accepted. If there are no requests made in a week, x_9 defaults to 0.5. A default of 0.5 is chosen over 1 since the latter would compensate for poor satisfaction in other criteria. This could essentially penalise drivers that do not file time-off requests, giving them worse schedules than they otherwise would have.

$$c^{(0)}_{9,d} = x_9$$

6.3.3 Schedule distribution

Consecutive shifts

Drivers often prefer their number of consecutive shifts x_{10} to be limited. Even if drivers do not have a strong preference on this criterion, the company's safety regulations still mandate a focus on it. Two shifts are considered consecutive if the driver's resting time between them is less than 24 hours.

The best-case threshold is five consecutive shifts, since lower numbers would only lead to an undesirable distribution of free days. The worst-case threshold is set to ten consecutive shifts. The criterion score is linear between these thresholds:

$$c^{(0)}_{10,d} = \lambda(x_{10}, 10, 5)$$

Consecutive free days

Drivers may prefer to have two consecutive free days per week, as opposed to two separate, single free days. In addition, it is very important that the drivers have proper free days at all. A single free day is defined by a resting time between 24 and 48 hours, whereas a double free day has a resting time of at least 48 hours. If the driver has a double free day, the key value x_{11} gets a score of 1. Otherwise, x_{11} is 0.25 times the number of single free days.

$$c^{(0)}_{11,d} = x_{11}$$

Resting time

Drivers may prefer that their resting time between shifts is longer than the minimum set by the safety regulations. This means, specifically, that the resting time should be distributed as evenly as possible throughout the week. The following formula is used to determine a penalty score for each resting time:

$$\rho = \begin{cases} (14 - r)^2 & \text{if } r < 14 \\ 0 & \text{if } r \geq 14 \end{cases}$$

Here, ρ is the penalty score and r is the resting time. As such, the score can vary between 9 for a resting time of 11 hours and 0 for a resting time of 14 hours or more.

The key value x_{12} for this criterion is the sum of all such penalty scores. The best-case threshold is a total penalty score of 0, whereas the worst-case threshold is a total score of 36.

$$c^{(0)}_{12,d} = \lambda(x_{12}, 36, 0)$$

6.4 Bi-objective optimisation

While robustness is intrinsically still about monetary costs, driver satisfaction is a different criterion altogether. The client wishes that multiple schedules are presented, each with a different trade-off between monetary costs and satisfaction, so the final decision can be made by the planning staff. This requires a way for the simulated annealing algorithm to optimise for both criteria at the same time, while still exploring solutions that prioritise one criterion over the other.

A great way of determining the trade-off schedules is through the use of a Pareto-optimal front. This is the set of non-dominated solutions, meaning that for all solutions in the front, neither criterion can be improved without the other worsening. The Pareto front contains all solutions worth considering for the planning staff.

An easy method of determining the Pareto front would be to run the simulated annealing algorithm multiple times, with differently weighted averages between the criteria or differing minimum thresholds on one of the criteria. This would, however, increase the runtime of the algorithm many times over, which is unnecessary.

A much faster alternative is to run the algorithm once, but measure the cost differences using a weighted average where the weights change every simulated annealing cycle. This way, all different trade-offs between the criteria are explored. We implement this using an *adjusted cost* $f'(\sigma)$ for each schedule σ calculated using the following formula:

$$f'(\sigma) = f(\sigma) * \Big(1 + \varphi\big(1 - s(\sigma)\big)\Big)$$

Here, $s(\sigma)$ is the schedule satisfaction score as defined in Section 6.1 and φ is the *satisfaction weight factor*. Each cycle, φ is set to a randomly chosen factor between 0 and 10. The simulated annealing algorithm considers the difference in adjusted cost when deciding whether to accept or reject neighbouring solutions.

While the algorithm is being executed in this way, the Pareto front of solutions is stored. This is done by storing, for each level of satisfaction, the best solution with a satisfaction equal to or higher than that level. The algorithm considers each one-percent interval to be a different level.

7

Experimental results

To be able to use and test the algorithm developed in this study, we developed an extensive C# prototype. Using this prototype, experimental analysis was performed with real-life instance data. This analysis is key to understanding the quality and speed of the algorithm, as well as how to best configure some of its parameters. This section will discuss the results of the analysis.

Used in the experiments are three real-world problem instances of the client, each of a different week. Table 7.1 shows the number of activities in each instance. The same real-world numbers are used for each instance of 17 internal drivers and 68 shifts from external drivers available for hire.

These experiments were performed on a machine with an Intel Core i7-7700HQ 2.80-gigahertz processor with 4 physical cores, 8 virtual cores and 16 gigabytes of RAM. The algorithm was run on 8 parallel threads.

7.1 Convergence

Simulated annealing is an algorithm heavily based on probability. There is no guarantee that an optimal solution is ever found, but merely an expectation that the algorithm yields high-quality results after some particular amount of time. The only method of determining this amount of time is by performing long experiments and examining the convergence of the solutions. Ideally, we would examine the quality of the entire Pareto front over time, but it is hard to represent this quality as a single value. Therefore, we view the lowest-cost and highest-satisfaction solutions found over time, which give a good indication of convergence.

Executing the algorithm for ten billion iterations showed a progression of the lowest-cost solution as shown in Figure 7.1 and of the highest-satisfaction solution in Figure 7.2. Both figures show the progress as a percentage relative to the final value of that instance.

Figure 7.1 shows that the algorithm reaches a lowest cost within 3% of the final value at around 1 billion iterations. The speed of convergence then differs heavily between the instances, taking between two and nine billion iterations to reach apparent convergence. Figure 7.2 shows that a highest satisfaction within a relative 3% of the final value is also reached after around 1 billion iterations. Here, too, the speed of convergence varies between instances, taking between four and nine billion iterations.

These statistics show that a decent solution can likely be found by executing the algorithm for 1 billion iterations. To achieve a seemingly converged solution, the algorithm requires around ten billion iterations. A good balance between runtime and quality could be achieved at around four billion iterations. These numbers present different options to

the client, between which they can choose depending on the amount of time available and the importance of a near-perfect solution.

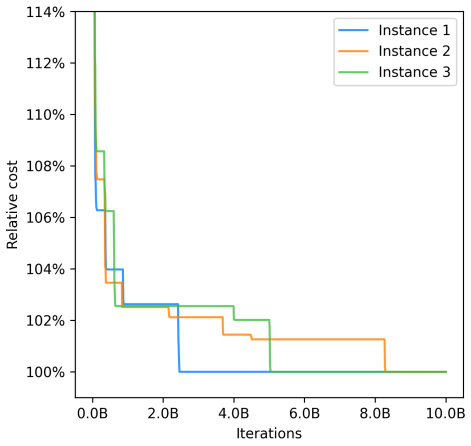


Figure 7.1: Best solution cost found over time by for each instance, relative to the final best solution cost, during a run of ten billion iterations.

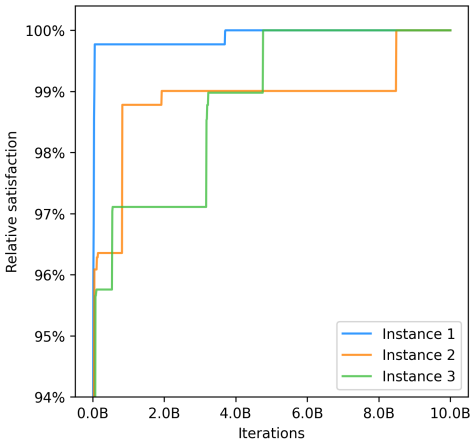


Figure 7.2: Best solution satisfaction found over time for each instance, relative to the final best solution satisfaction, during a run of ten billion iterations.

The differences between these recommended runtimes can be further examined in Figure 7.3, which shows the Pareto front after each of the three execution lengths. The figure shows that for instances 2 and 3, the differences between the different execution lengths are rather minor. For instance 1, the run of one billion iterations gives a significantly worse Pareto front, but only in the lower-cost, lower-satisfaction part of the front. All in all, the differences are noticeable but relatively small.

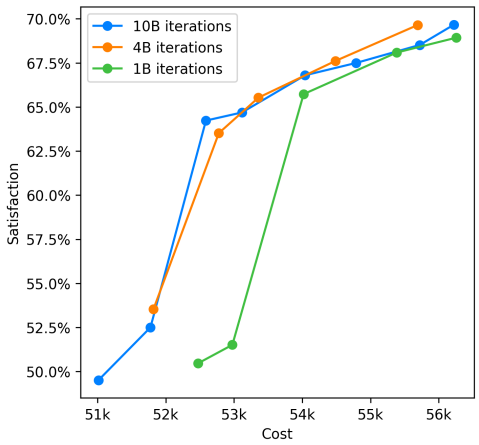
7.2 Runtime

The complete run of ten billion iterations was completed in 265 minutes or about 4.5 hours. This means that the average speed of the algorithm was 630 thousand iterations per second. A run of one billion iterations takes approximately 27 minutes to execute, with one of four billion iterations taking about 106 minutes. The latter two runtimes are very acceptable when considering that in the current situation, the planning staff spends two full working days actively creating each weekly schedule. The algorithm is not only finished much sooner, but no human involvement is required during the execution.

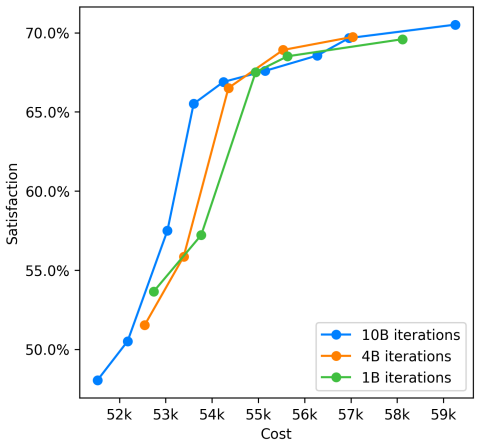
7.3 Quality

The most important metric of the algorithm's performance is the quality of its solutions. Table 7.1 also shows, for each instance, the runtime and solutions of a run with four billion iterations. Each of these solutions is a Pareto front of all schedules found by the algorithm, based on their cost and satisfaction. The robustness cost of each schedule is also listed separately. The Pareto fronts are also shown visually as the orange lines in Figure 7.3.

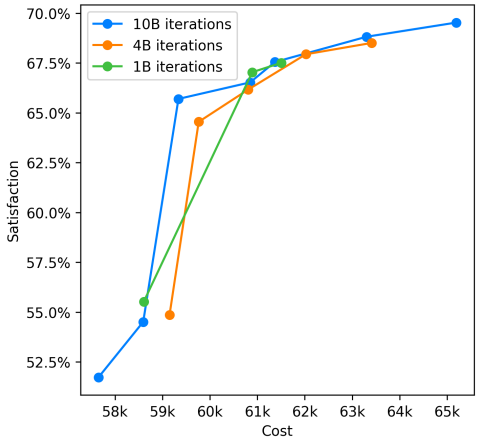
It is hard to evaluate the objective quality of these solutions, since there are no benchmarks available for this problem. However, they can be compared with the current, human-made schedules. For instances 1 and 2, the planning staff did not manage to find a solution



(a) Instance 1



(b) Instance 2



(c) Instance 3

Figure 7.3: Visualisations of the found Pareto fronts for each problem instance after runs of one billion, four billion and ten billion iterations.

where all activities are scheduled, but the algorithm did. This shows that the schedules generated by the algorithm are highly effective.

For instance 3, the planning staff found a complete solution too. The available data about their schedule, however, is imperfect, since it contains many overlapping activities and safety constraint violations. In our best estimate, the company's schedule has a cost of about 61.000 and a satisfaction of 31%. However, due to the overlapping activities in this schedule, we assume the real cost would have been considerably higher. Still, this cost estimate is comparable to that of the algorithm's fully feasible solutions. Meanwhile, the estimated driver satisfaction is much lower than that of the algorithm's solutions. These comparisons show that the algorithm provides schedules that are likely cheaper, more satisfying for drivers, and more adherent to the company's rules.

The algorithm's solutions have also been examined by the client's planning staff, who have confirmed that they obey all constraints. Additionally, they deemed most of the shifts in each schedule to be logical and feasible. A small share of shifts may need to be edited, because they have long waiting or travel times that, while allowed, are not very practical. All in all, the generated schedules provide a good foundation to base final schedules on, which is precisely the goal defined in Chapter 2.

Table 7.1: Details of the problem instances and their found solutions. Given for each schedule in a Pareto front are its total cost, its robustness cost and its satisfaction. Note that the total cost consists of the theoretical monetary cost and the robustness cost.

Instance	#Activities	Runtime (s)	Pareto front		
			Total cost	Rob. cost	Satisfaction
1	328	6290	51820	3013	53.53%
			52778	3028	63.52%
			53358	2525	65.53%
			54485	2836	67.61%
			55692	3033	69.65%
2	310	6437	52544	2838	51.52%
			53388	3101	55.86%
			54355	3181	66.51%
			55535	2655	68.91%
			57041	3178	69.74%
3	368	6676	59149	4864	54.86%
			59764	4096	64.55%
			60808	4198	66.17%
			62022	3968	67.94%
			63412	3057	68.51%

7.4 Practical advantages

The experimental analysis shows that the algorithm can improve the client's current planning process in several ways. Firstly, it should reduce the workload of the planning staff. The algorithm should provide them with a good basis on which to build their final schedules, removing a considerable part of the work. Secondly, it should help create schedules that improve the drivers' satisfaction and detail the trade-off between costs and satisfaction. Lastly and most crucially, the algorithm should allow the client to increase their profits, since its efficiency allows both for more activities to be performed and for all shifts to be scheduled with higher cost-efficiency.

8

Conclusion

In this study, we presented a simulated annealing algorithm to assist planning staff in creating weekly schedules for cargo train drivers. These schedules must consist of valid sets of shifts that together cover all activities. The algorithm provides schedules with different trade-offs between monetary costs and driver satisfaction, which the planning staff can use to construct a final schedule. Included in the monetary costs are expected additional expenses from delays, calculated using a robustness model based on consecutive activities. Driver satisfaction is calculated as an aggregate of twelve satisfaction criteria per driver, with a possibility of having different preferences for each driver. As its solution, the algorithm presents a Pareto-optimal front of schedules representing the costs-satisfaction tradeoff, from which the planning staff can select the best option.

The algorithm takes into account many real-world constraints like maximum shift lengths, minimum resting times, contract hours, and driver qualifications. It works with heterogeneous drivers that live in different locations and therefore have different travel times that often require compensation. Additionally, the system considers both internal drivers of the company and external drivers available for hire, each type having different salary rates that vary throughout the day and week.

Experimental analysis with real-world data shows that the algorithm provides feasible schedules in acceptable runtimes. These schedules are significantly more effective than the current human-made schedules, giving lower costs, higher driver satisfaction and higher robustness. The algorithm can therefore strongly reduce the workload for the client's planning staff, while increasing both the profits of the client and the satisfaction of their drivers.

8.1 Future research

While the methods used in this study give good results, there is certainly room to improve them further. It would be worthwhile to enhance the developed algorithm further, such that it could eventually produce schedules ready for immediate use without requiring human adjustments. This would require both further improvements to the current facets of the algorithm and the addition of more components that could not be taken into account in this study.

Firstly, the robustness calculation of the algorithm could be further extended. Activities could be split into multiple risk categories with different probability distributions for their delays. Moreover, approaches could be considered to base the robustness calculation on longer sequences of activities than just consecutive ones.

Secondly, further improvements could be made to the driver satisfaction calculation. The method of aggregating satisfaction criteria to an eventual schedule satisfaction score

could be further improved to make the score more representative of the real-world satisfaction. Additionally, more criteria could be added. For example, travel time within shifts could be minimised, instead of only minimising travel to and from home. Likewise, a minimisation of waiting time might be desirable for drivers. Lastly, drivers may prefer a high degree of variation in the types of locomotives they drive.

Thirdly, several constraints could be changed to reflect the real-world situation more accurately. For example, while the planned start and end times of activities are considered fixed in this study, they have varying degrees of flexibility in reality. Allowing for activities to be slightly shifted can allow for even more effective scheduling. Additionally, labour laws generally allow for slight violations of the rules, as long as these happen rarely. It would therefore be useful to show solutions with small constraint violations. Furthermore, stand-by shifts are in practice added to schedules to reduce the effects of operational disruptions, thereby increasing robustness. Including these in the algorithm would be useful.

Lastly, additional improvements could be considered to the simulated annealing algorithm. These could include the introduction of new neighbourhood operations, as well as improvement to the current operations like a weighted selection of trips and drivers to prioritise those most likely to improve the solution.

On a different note, there are additional use cases for variants of the current algorithm. One example is a variant that can update an initial schedule with new information after the planning deadline. Since changing driver schedules after this deadline requires monetary compensation for the drivers, this algorithm should change as few activities as possible. Another variant is one to help solve operational disruptions, again without incurring too many costs due to changed schedules.

Furthermore, there are also interesting topics for future studies in the broader field of research. While this study focused on the situation of the client, Rail Force One, many parts of the developed algorithm could likely also be applied to related problems. For example, we assume the robustness approach of this paper to function well for applications of problems like Multiple-Depot Vehicle Scheduling, Bus Driver Scheduling and Airline Crew Scheduling. Applications of the latter two problems would likely also be suited for the multiple-criteria-based satisfaction approach from this study.

Bibliography

Al-Fawzan, M. A., & Haouari, M. (2005). A bi-objective model for robust resource-constrained project scheduling. *International Journal of production economics*, *96*(2), 175–187.

Alfieri, A., Kroon, L., & Van de Velde, S. (2007). Personnel scheduling in a complex logistic system: A railway application case. *Journal of Intelligent Manufacturing*, *18*(2), 223–232.

Antunes, D., Vaze, V., & Antunes, A. P. (2019). A robust pairing model for airline crew scheduling. *Transportation Science*, *53*(6), 1751–1771.

Bayer, D. A. (2012). *Pairing generation for airline crew scheduling* (Master's thesis). University of Waterloo.

Borndörfer, R., Schelten, U., Schlechte, T., & Weider, S. (2006). A column generation approach to airline crew scheduling. In *Operations research proceedings 2005* (pp. 343–348). Springer.

Chtourou, H., & Haouari, M. (2008). A two-stage-priority-rule-based algorithm for robust resource-constrained project scheduling. *Computers & industrial engineering*, *55*(1), 183–194.

Desaulniers, G., Desrosiers, J., & Solomon, M. M. (2002). Accelerating strategies in column generation methods for vehicle routing and crew scheduling problems. In *Essays and surveys in metaheuristics* (pp. 309–324). Springer.

Diepen, G., van den Akker, J. M., Hoogeveen, J. A., & Smeltink, J. W. (2012). Finding a robust assignment of flights to gates at amsterdam airport schiphol. *Journal of Scheduling*, *15*(6), 703–715.

Domínguez-Martín, B., Rodríguez-Martín, I., & Salazar-González, J.-J. (2018). The driver and vehicle routing problem. *Computers & Operations Research*, *92*, 56–64.

Emden-Weinert, T., & Proksch, M. (1999). Best practice simulated annealing for the airline crew scheduling problem. *Journal of Heuristics*, *5*(4), 419–436.

Fischetti, M., & Monaci, M. (2009). Light robustness. In *Robust and online large-scale optimization* (pp. 61–84). Springer.

Fores, S. (1996). *Column generation approaches to bus driver scheduling* (Doctoral dissertation). University of Leeds.

Gustafsson, T. (1999). A heuristic approach to column generation for airline crew scheduling. Citeseer.

Hadjar, A., Marcotte, O., & Soumis, F. (2006). A branch-and-cut algorithm for the multiple depot vehicle scheduling problem. *Operations Research*, *54*(1), 130–149.

Hanafi, R., & Kozan, E. (2014). A hybrid constructive heuristic and simulated annealing for railway crew scheduling. *Computers & Industrial Engineering*, *70*, 11–19.

Jorge Leon, V., David Wu, S., & Storer, R. H. (1994). Robustness measures and robust scheduling for job shops. *IIE transactions*, *26*(5), 32–43.

Jütte, S., Müller, D., & Thonemann, U. W. (2017). Optimizing railway crew schedules with fairness preferences. *Journal of Scheduling*, *20*(1), 43–55.

Khemakhem, M. A., & Chtourou, H. (2013). Efficient robustness measures for the resource-constrained project scheduling problem. *International Journal of Industrial and Systems Engineering*, *14*(2), 245–267.

Kobylański, P., & Kuchta, D. (2007). A note on the paper by ma al-fawzan and m. haouari about a bi-objective problem for robust resource-constrained project scheduling. *International Journal of Production Economics*, *107*(2), 496–501.

Kulkarni, S., Krishnamoorthy, M., Ranade, A., Ernst, A. T., & Patil, R. (2018). A new formulation and a column generation-based heuristic for the multiple depot vehicle scheduling problem. *Transportation Research Part B: Methodological*, *118*, 457–487.

Kwan, R. S., & Kwan, A. (2007). Effective search space control for large and/or complex driver scheduling problems. *Annals of Operations Research*, *155*(1), 417–435.

Laplagne, I. E. (2008). *Train driver scheduling with windows of relief opportunities* (Doctoral dissertation). University of Leeds.

Liebchen, C., Lübbecke, M., Möhring, R., & Stiller, S. (2009). The concept of recoverable robustness, linear programming recovery, and railway applications. In *Robust and online large-scale optimization* (pp. 1–27). Springer.

Lim, A., & Zhu, W. (2006). A fast and effective insertion algorithm for multi-depot vehicle routing problem with fixed distribution of vehicles and a new simulated annealing approach. *International conference on industrial, engineering and other applications of applied intelligent systems*, 282–291.

Lourenço, H. R., Paixão, J. P., & Portugal, R. (2001). Multiobjective metaheuristics for the bus driver scheduling problem. *Transportation science*, *35*(3), 331–343.

Lučic, P., & Teodorovic, D. (1999). Simulated annealing for the multi-objective aircrew rostering problem. *Transportation Research Part A: Policy and Practice*, *33*(1), 19–45.

Meijer, M., Huisman, D., & van Dijk, W. (2017). Nonlinear variation constraints in railway crew scheduling.

Moudani, W. E., Cosenza, C. A. N., Coligny, M. d., & Mora-Camino, F. (2001). A bi-criterion approach for the airlines crew rostering problem. *International Conference on Evolutionary Multi-Criterion Optimization*, 486–500.

Passage, G., Hoogeveen, H., & van den Akker, M. (2016). *Combining local search and heuristics for solving robust parallel machine scheduling* [Unpublished master's thesis]. Utrecht University. https://dspace.library.uu.nl/handle/1874/334269

Peng, K., Shen, Y., & Li, J. (2015). A multi-objective simulated annealing for bus driver rostering. *Bio-Inspired Computing-Theories and Applications*, 315–330.

Pepin, A.-S., Desaulniers, G., Hertz, A., & Huisman, D. (2009). A comparison of five heuristics for the multiple depot vehicle scheduling problem. *Journal of scheduling*, *12*(1), 17–30.

Porokka, A., et al. (2017). *Train driver rostering in finland considering driver satisfaction* (Master's thesis). Aalto University.

Quesnel, F., Desaulniers, G., & Soumis, F. (2020). Improving air crew rostering by considering crew preferences in the crew pairing problem. *Transportation Science*, *54*(1), 97–114.

Renaud, J., Laporte, G., & Boctor, F. F. (1996). A tabu search heuristic for the multi-depot vehicle routing problem. *Computers & Operations Research*, *23*(3), 229–235.

Ribeiro, C. C., & Soumis, F. (1994). A column generation approach to the multiple-depot vehicle scheduling problem. *Operations research*, *42*(1), 41–52.

Schöbel, A. (2014). Generalized light robustness and the trade-off between robustness and nominal quality. *Mathematical Methods of Operations Research*, *80*(2), 161–191.

Shen, Y. (2001). *Tabu search for bus and train driver scheduling with time windows* (Doctoral dissertation). University of Leeds.

van den Broek, R., Hoogeveen, H., & van den Akker, M. (2018). How to measure the robustness of shunting plans. *18th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2018)*.

van Kooten Niekerk, M. E., van den Akker, J. M., & Hoogeveen, J. A. (2017). Scheduling electric vehicles. *Public Transport*, *9*(1), 155–176.

van Strien, F. (2021). *Locomotive assignment with new traction types for rail force one* [Unpublished master's thesis]. Utrecht University.

Wang, C., Guo, C., & Zuo, X. (2021). Solving multi-depot electric vehicle scheduling problem by column generation and genetic algorithm. *Applied Soft Computing*, *112*, 107774.

Wen, M., Linde, E., Ropke, S., Mirchandani, P., & Larsen, A. (2016). An adaptive large neighborhood search heuristic for the electric vehicle scheduling problem. *Computers & Operations Research*, *76*, 73–83.

Wilson, M., Klos, T., Witteveen, C., & Huisman, B. (2014). Flexibility and decoupling in simple temporal networks. *Artificial Intelligence*, *214*, 26–44.

Yen, J. W., & Birge, J. R. (2006). A stochastic programming approach to the airline crew scheduling problem. *Transportation Science*, *40*(1), 3–14.

Zhou, S.-Z., Zhan, Z.-H., Chen, Z.-G., Kwong, S., & Zhang, J. (2020). A multi-objective ant colony system algorithm for airline crew rostering problem with fairness and satisfaction. *IEEE Transactions on Intelligent Transportation Systems*, *22*(11), 6784–6798.