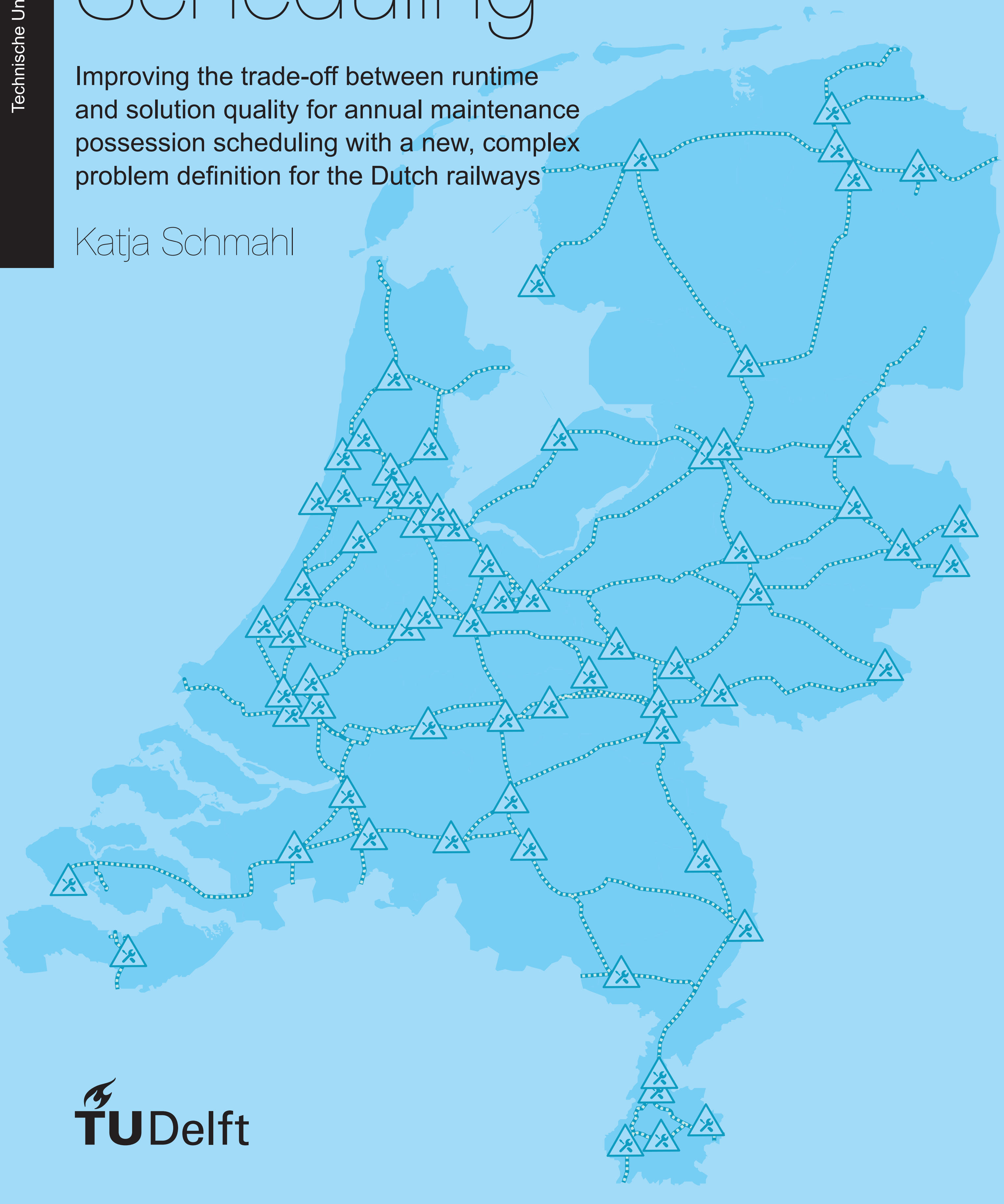
Railway Maintenance Scheduling

Improving the trade-off between runtime and solution quality for annual maintenance possession scheduling with a new, complex problem definition for the Dutch railways

Katja Schmahl



TUDelft

Railway Maintenance Scheduling

Improving the trade-off between runtime and solution quality for annual maintenance possession scheduling with a new, complex problem definition for the Dutch railways

by

Katja Schmahl

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Monday October 30, 2023 at 11:00 AM.

Student number:	4726022	
Project duration:	Feb 1, 2023 – Oct 30, 2023	
Thesis committee:	Dr. ir. N. Yorke-Smith,	TU Delft, supervisor
	Dr. ir. J. A. Baaijens,	TU Delft
	Dr. ir. H. J. Griffioen,	TU Delft
	Ir. M. Oudshoorn,	Macomi B.V.

TU Delft

Abstract

There is increasingly more expensive maintenance that needs to be performed on the Dutch railway network. Good maintenance schedules reduce costs, minimise hindrance to passenger and freight travel, and follow restrictions imposed by available resources, legislation and other agreements. The railway maintainer has modelled this maintenance scheduling problem, among others, on an annual level. This problem definition is very precise with different conflicting non-linear constraints and cost parts. The demands of the solving method depend on the phase of the scheduling process. In early phases, low runtime is most important, whereas best solution quality outweighs this runtime when the maintenance work is more finalised. In 2019, a simple greedy algorithm found good solutions fast, and a hybrid greedy-evolutionary algorithm was developed that resulted in the best schedules. However, since then, the problem definition has been made more realistic and thus complex. Therefore, this hybrid greedy-evolutionary algorithm is no longer feasible, and creating a maintenance schedule takes significantly longer than before. Research is necessary to better understand the impact of the more realistic model, and to once more have a good trade-off between solving time and solution quality available for the schedulers. In this thesis, we aim to achieve this by improving different aspects of the problem and solving methods. Most experiments were done with the maintenance schedule of 2024, and results were verified on the years of 2023 and 2025. First, general problem analysis and implementation improvements reduced the runtime from around twenty-four hours to three hours with the greedy algorithm. Then, approximations were applied in the passenger hinder to further reduce the runtime by around half with a negligible negative impact on the solution quality. Due to these speed-ups, it was possible to use more elaborate solving methods. New experimental results showed that the greedy algorithm still finds solutions fast. The hybrid greedy-evolutionary algorithm found better quality schedules, but required more runtime. Furthermore, a novel solving method with look-aheads was proposed, which showed some potential for cost reductions, but was dominated by the hybrid algorithm. Every algorithm uses the greedy heuristic as a subroutine. Results showed the importance of finding the right order for greedily scheduling the requests. A proposed new order function improved the quality of the resulting maintenance schedules even further. To conclude, the increase in complexity of the problem definition in recent years has made solving more difficult. To still find good solutions in a similar time, a better performing greedy heuristic was necessary. By applying different runtime optimisations to the objective evaluation and improving the solution quality of the greedy heuristic, a good trade-off between runtime and solution quality, using different solving methods, was realised for creating annual maintenance schedules for the Dutch railway network.

Preface

Before you lies the result of my thesis project for the master of Computer Science at the Delft University of Technology. This thesis outlines the research results, made possible through an internship at Macomi. I originally chose to conduct my thesis at a company, because I wanted to work with a real-world problem. If only I truly realised what this entailed beforehand. During the thesis project, I ran into numerous advantages and disadvantages of exactly this decision. I ended up needing to change the direction of my research multiple times. Despite the fact that it was frustrating at times, I have learned a lot and working with a real-world problem helped keep me motivated. Even if it was just so I would be delayed by maintenance on my own travels less.

I would like to thank my supervisor from the TU Delft, dr. ir. Neil Yorke-Smith, for connecting me with the internship possibility at Macomi, and for always being available to provide constructive feedback and answer my many questions. I want to thank everyone at Macomi for making me feel welcome in your offices. I especially would like to express my gratitude to both Menno Oudshoorn and Timo Koppenberg, who provided me with guidance and useful ideas throughout this project. Finally, I would like to thank my family and friends, for all the support and some much-needed distraction.

Katja Schmahl
Delft, October 2023

Contents

1 Introduction

The Dutch railway infrastructure must be maintained to allow passenger and freight traffic to continue safely. The infrastructure maintenance and expansion is the responsibility of ProRail, a Dutch government agency. ProRail determines both what maintenance needs to be done, and when that maintenance will be performed. For safety reasons, no trains are allowed to traverse the sections of the network undergoing maintenance. The amount of required maintenance is too large to do it in the train-free hours of the night. As such, trains will have to be hindered regularly. ProRail attempts to schedule the maintenance works in a manner that optimises the triangle of reliability, availability, and affordability. Scheduling the maintenance work is done on different levels, ranging from daily schedules to multi-year schedules. In this thesis, research is conducted on the creation of the yearly schedule, which contains the larger hindering projects. A precise objective has been defined for this by ProRail. The current scheduling methods are insufficiently adapted to this objective. Therefore, the problem and existing solving methods are researched and improved upon in this thesis.

1.1 Motivation

Trains are an integral part of the public transportation system in the Netherlands. ProRail is responsible for the maintenance of more than seven thousand kilometres of rail [1]. The railway network can be seen in fig. 1.1. Upon this railway, 157 million train kilometres were driven in 2022 [1]. The Dutch railway network is the most intensively used of the European Regulator Group, as can be seen in fig. 1.2. Usage is only expected to increase in the future [1]. This higher usage makes scheduling more difficult, since there are fewer periods without trains during which maintenance can be scheduled without hinder. Operating with more or larger trains also means that the degradation of the railway infrastructure is expected to accelerate. The amount of maintenance required will only increase over time. It increased with around 20% in 2022 compared to 2021, this increase is expected to continue towards future years [1].

Increased availability of public transport is a current and important point on the Dutch political agenda, both for sustainability and accessibility reasons [2]. Therefore, multiple investments are planned to allow higher frequencies, for national and international trains. To implement these expansions, train traffic will also inherently have to be hindered.

Minimising the costs of, and the hinder caused by, all infrastructure projects is therefore increasingly important. This allows us to keep the public transport safe and available on both the short and long term. Better schedules have two main advantages. The first advantage is financial: scheduling maintenance intelligently can reduce the costs of maintenance projects, by for example reducing the personnel costs. The second advantage is societal: better maintenance scheduling can reduce hindrance to passenger and freight trains. Good schedules adhere to all rules set by the government, and agreements made with other parties. Better solving methods could also save schedulers time. Moreover, it allows schedulers to consider multiple different scenarios with all concerned parties.

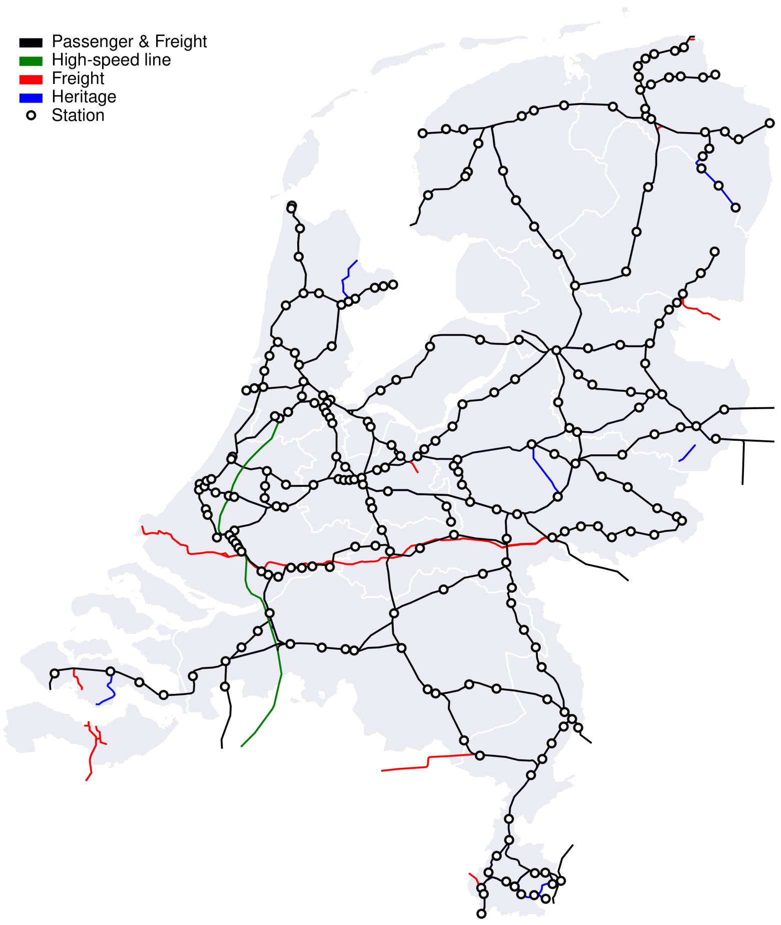


Figure 1.1: The railway infrastructure in the Netherlands. [3]

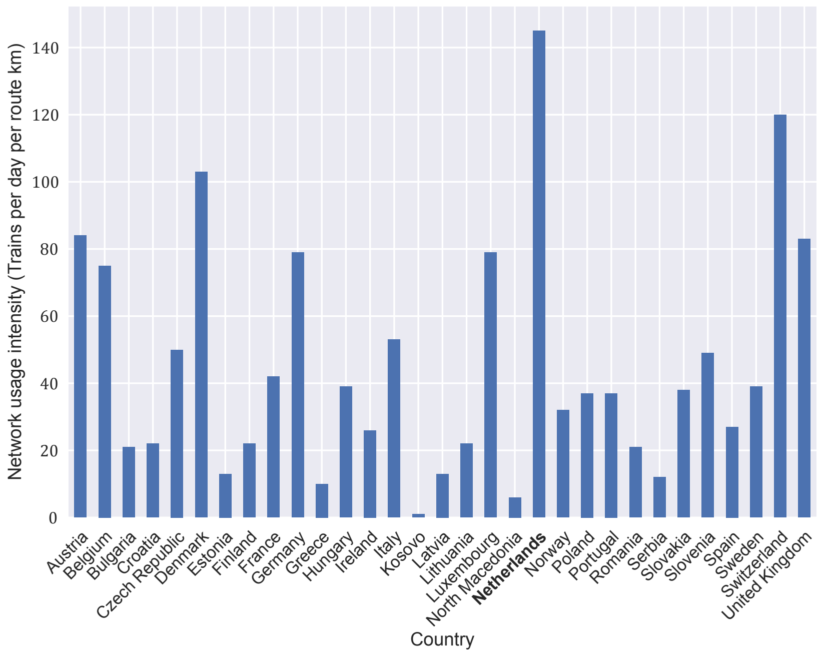


Figure 1.2: The network usage intensity of countries in the *Independent Regulator's Group-Rail* [4]. The Netherlands has the highest usage intensity with 145 trains per day per route km.

Besides improving the resulting schedules, more research could provide improved understanding of the problem and its possible solving methods. Better problem understanding could have a few advantages. Difficult decisions in the maintenance management, such as prioritising which maintenance should be done most urgently or how to (re-)train personnel, have to be made continuously. Having insight in how these potential changes will impact the total amount of disruption, and the feasibility of getting all required work done while keeping to the specified constraints, could support better decision-making. Similarly, this understanding can aid in (political) discussions on the future of railway transport in the Netherlands. Also, the advantages and drawbacks of this realistic cost and constraints modelling can be investigated by analysing the impact of the current increase in model complexity. This could help to change the model in the future, and provide insight that could improve both solving and modelling of similar problems.

1.2 Problem statement

The Dutch railway manager (ProRail) decides on a set of maintenance projects per year, that should be executed within predetermined time windows. It is necessary to create schedules which determine what infrastructure maintenance will be performed when. To find the best possible maintenance schedule, the problem is defined precisely, with a complex set of constraints and a precise objective function.

The scheduling problem is very high-dimensional, with hundreds of maintenance projects that need to be scheduled. Many aspects of the problem objective are defined on a high level of granularity. For example, passenger travel data is defined on an hourly level. Hindrance needs to be calculated for every frequently travelled origin-destination pair, for every hour in the year-long schedule. Besides that, there is a large amount of diverse constraint types, which introduces non-linearity into the objective. Different cost parts and constraints are conflicting. For example, scheduling during the day means that labour is cheaper, whereas scheduling at night hinders fewer passengers.

At ProRail, the final schedule is evaluated and made by a team of schedulers. Their demands have been captured in the objective. This is however never completely representative of what is considered most important. ProRail has partnered with Macomi to create a tool for finding good candidate schedules of the defined set of required train-free periods. Macomi is a company specialised in simulation and optimisation. It is important to note that throughout this scheduling process, the maintenance work is changing and becoming more specific. As such, the requirements of this tool are also dynamic. In earlier phases of the scheduling, the input is still less detailed, and solutions need not be optimised completely. When new projects or different configurations are considered, for example, rapidly determining how this impacts the scheduling feasibility is more useful. When the input is (almost) finalised, solving time is not nearly as important, but the schedulers care more about finding the very best solutions.

A few years ago, a fast greedy algorithm and a specialised evolutionary algorithm were developed for finding good schedules, which were successfully combined into a hybrid greedy-evolutionary technique [5]. However, the constraints and costs have been substantially altered since then, to more accurately represent reality. This new objective is more computationally expensive to evaluate, making it difficult to effectively go through the search space. The developed greedy algorithm, which cost about fifteen minutes before, now takes over twenty-four hours. Because of this, the previously preferred hybrid version is no longer currently in use, and designing good schedules has become more time-consuming. Therefore, the impact of these changes to the problem definition on the solving methods needs to be further researched.

1.3 Research goal and questions

The final goal of this thesis is to research and improve the solving of the maintenance scheduling problem for the Dutch railways. Improvement would either be shorter runtime, better solution quality, or preferably both. This would ideally make it possible to choose the current algorithm settings based on the runtime-quality trade-off preference at that moment in the scheduling phase. Besides, better understanding of this trade-off is another research goal, as it can be helpful for potential future changes to the problem.

The main research question that will be answered in this thesis is: **“How can improving different problem and solution aspects of the yearly maintenance scheduling on the Dutch railway network help to obtain better understanding and improved usability of the trade-off between runtime and solution quality?”** This will be done by first answering each of the following sub-questions. These answers will finally be combined to answer the main research question.

1. How do different aspects of the problem and the currently used algorithms influence the runtime and solution quality?
2. How can approximation of passenger detour paths be used for faster solving?
3. How do different search strategies solve the scheduling problem in terms of solving time and solution quality?
4. How can new prioritisation techniques be applied to improve the solution quality?
5. How do techniques that improve the available trade-off between the runtime and solution quality for the schedule of 2024 generalise to different years of input data?

1.4 Anonymity

All problem input data used in this thesis is based on real maintenance required for the Dutch railway network, and the real passenger and railway data. As these maintenance costs are considered confidential, all objective values have been divided by a large constant, which is not given. As such, cost improvements which may seem small in these anonymous values can still be very substantial improvements for ProRail.

1.5 Structure of this document

The structure of the remainder of this thesis is as follows. First the background chapters 2 and 3, then the content chapters 4 to 8 which aim to answer the research sub-questions, and finally the conclusion in chapter 9.

Chapter 2 introduces the necessary related work. More background is given in the content chapters where it is used. Chapter 3 provides a detailed description of the problem, as well as an analysis of the most important characteristics of the used input.

Chapter 4 will explain the current algorithms for finding and evaluating solutions, and some first improvements will be introduced. Chapter 5 will apply approximation to an important aspect of the evaluation to reduce runtime. Chapter 6 shows how different search strategies allow trading off runtime and solution quality. Chapter 7 will propose a new technique that is designed to improve the solution quality obtained by each of the available search strategies. Chapter 8 will show how the results of all other content chapters apply to different scheduling years, to analyse the robustness of the solutions.

Finally, in chapter 9, all conclusions from the content chapters are summarised and the main research question is answered. Also, all future research directions are summarised.

2
Background and Related Work

In this chapter, the necessary background for this thesis is provided. First, an overview of research relating railway maintenance scheduling on the Dutch railway is given in section 2.1. Then, an overview of other related research in the field of railway maintenance scheduling is provided in section 2.2. Finally, in section 2.3, the identified research gap that this thesis aims to fill is explained.

2.1 Previous work on the Dutch railway maintenance scheduling

The most important work is the thesis that this research most directly builds upon by Oudshoorn [5], and the accompanying paper by Oudshoorn, Koppenberg, and Yorke-Smith [6]. This contains a comparison of solving methods on the yearly scheduling problem. This work used an earlier version of the problem definition, with a simpler solution evaluation. In this research, a greedy constructive algorithm and a specialised evolutionary algorithm were designed. These algorithms were successful in reducing different aspects of the cost. Therefore, the two algorithms were combined into a hybrid greedy-evolutionary, which was the best solving method found for the problem. Some algorithms in this work are still applicable, and these will be explained further in chapters 4 and 6. Besides, a multi-objective approach was also tested in this work; numerical results showed this was unsuitable for this problem. A simulated annealing algorithm was also considered, but this was discarded as it was outperformed by the evolutionary algorithm.

In 2011, a master thesis research was conducted on the annual Dutch maintenance schedule [7]. In this work, the 10 most important maintenance activities were considered, and a case study was presented for the maintenance in one railway yard.

Recently, another master thesis, by Weert [8], was written on the Dutch railway scheduling. This work researched the effects of incorporating events into the scheduling. The main conclusion of this work was that an optimal solution could be found for creating schedules that minimise the passenger hindrance, with a proposed method to add more flexibility. They used a mixed-integer linear programming model of the problem, which was solved using a commercial optimiser.

Compared to the problem definition used in this thesis, they used a smaller scale and a more simplified objective. Weert [8] included a subset of the constraints that will be used in this work, and it seeks to optimise only the passenger hindrance. So, it does not take into account other aspects such as the personnel costs. Moreover, the passenger hindrance was computed using the assumption that passengers always travel on the shortest of the pre-computed k-shortest paths which is not blocked by maintenance. The hindrance in this thesis is instead computed using a fully dynamic shortest path. A case study was done for a schedule of 90 days using part of the Dutch railway network, as opposed to the annual schedule using the full network in this thesis.

Hertog, Zante-de Fokkert, Sjamaar, *et al.* [9] suggested a division of the Dutch railway network in working zones, for safety. No trains can be operated in the same zone where maintenance is being performed. Zante–de Fokkert, Hertog, Berg, *et al.* [10] used these working zones to determine sets of zones that can be blocked at the same time. They then designed a model for creating a weekly repeating maintenance schedule for preventive maintenance that can be done at night. These weekly schedules are used by ProRail for smaller maintenance, the larger maintenance is scheduled on a longer time span. Nijland, Gkiotsalitis, and Berkum [11] later proposed a novel mixed-integer linear programming (MILP) for the weekly schedules that considers the trade-off between hindrance for train operators and the maintenance management. A multi-objective formulation of the problem, considering contractor flexibility and train operator hindrance, was more recently proposed for this same problem on Dutch railway maintenance scheduling by [12], [13].

Budai-Balke [14] studied the problem of clustering preventive maintenance possession for small routine tasks and larger projects together, with the goal to minimise possession and maintenance cost. Pouryousef, Teixeira, and Sussman [15] developed the model further by considering multiple segments, improved handling of frequencies on routine tasks and traffic restrictions with penalties.

Besides the optimisation of railway infrastructure maintenance schedules, some recent work was done on the effects of the maintenance. Bešinović, Widarno, and Goverde [16] introduced an exact model for generating an alternative timetable in case of maintenance. An optimal alternative timetable is created for both freight and passenger trains, with a limited impact on the original timetable. Trepat Borecka and Bešinović [17] introduced the Multimodal Alternative Services for Possessions (MASP) problem, which supports the planning of alternative services when maintenance blocks the regular train service. The framework they developed provides schedules, passenger flow routing, and routes for alternative services (such as buses).

2.2 Other railway maintenance scheduling work

This thesis is part of an extensive body of research in the field of railway track maintenance planning and scheduling (RTMP&S). A categorisation and survey for problems in this category was given in Lidén [18] (and more elaborately in Lidén [19]). The field was split in three different levels: strategical, tactical and operational. The strategical level considers time spans of one to several years with larger strategic problems, such as organisation. Tactical problems are on a medium long time span, of weeks to years, and includes scheduling and timetabling. The operational category contains short-time problems relating to the final implementation. In this thesis, the problem is defined on a tactical level. Each of these levels were split into different classes. For the tactical level, these classes are: possession scheduling, rescheduling, and maintenance vehicle and team routing. The problem from this thesis is a possession scheduling problem. Possession scheduling was split up in four subclasses: major possession scheduling, regular possession pattern construction, possession and work coordination, and timetable compression. This thesis deals with a major possession scheduling problem.

Sedghi, Kauppila, Bergquist, *et al.* [20] expanded on this by covering the work done from 2015 to 2020. It also expanded the survey by developing a taxonomy with more attributes, and analysing the research trends and gaps. They recognise a critical role for possession scheduling in RTMP&S. Within the work in the class of possession scheduling, three main directions were identified. The first creates a train timetable, and then schedules maintenance in the train-free periods. The second fixes the maintenance schedule, and then creates a timetable for the trains in the periods without maintenance. The final direction is to schedule the two simultaneously. The first category was the most common approach, whereas the third category is receiving increased interest in recent years. This thesis falls within the second category, where the timetabling is done separately later.

Another trend that was identified is the increase in work considering condition-based maintenance, rather than predetermined maintenance. Condition-based maintenance results in more complex problem design, and is not done by ProRail on the yearly schedule level.

This work also identified the importance of creating a realistic estimation of the maintenance cost. Simplified objectives may result in an inadequate analysis of the RTMP&S strategies and decisions, as decision-makers often need to consider multiple conflicting objectives.

All work from this review, which falls in the same category as this thesis: possession scheduling, will briefly be summarised. The papers referenced in this taxonomy in the possession scheduling class with a network component level all used (mixed) integer programming. Solving was most often done using commercial solvers, or otherwise various heuristics and metaheuristics were used.

In 1999, Cheung, Chow, Hui, *et al.* [21] developed a constraint programming model for weekly track possession scheduling for the Hong Kong subway. The maintenance jobs are prioritised based on the content, and are to be scheduled when no train traffic is taking place.

Peng, Kang, Li, *et al.* [22] try to minimise travel and penalty costs, and assigns a maintenance team and time to each maintenance project. It schedules 333 projects in a year. A weekly precision is used. The cost is based on the travel time of the teams, and penalties for the defined soft constraints. This problem was extended in Peng and Ouyang [23], where more constraints were introduced and the solving method was improved. An initial solution is obtained from a relaxed problem, which is then improved with a parallel randomised local search.

Boland, Kalinowski, Waterer, *et al.* [24], and the accompanying paper [25], studied the problem of adjusting a maintenance plan for a coal chain railway. The maintenance tasks should be scheduled to maximise transportation throughput, while minimising the amount of changes from the original schedule. Each maintenance task can be moved up to a week, with a precision of half an hour. To make this problem tractable, multiple heuristics were applied to decrease the problem size and more efficiently solve the maximum flow problem.

Forsgren, Aronsson, and Gestrelius [26] proposed a mixed-integer programming (MIP) model to optimise train timetables and maintenance schedules together. Existing maintenance activities can be moved within pre-defined time windows. This model includes moving scheduled trains to a different time, rerouting trains, or cancelling trains. The total delay of the trains and the number of cancelled trains are minimised. It is applied in two realistic scenarios, which use a part of the Swedish railway network and a few track possessions.

Khalouli, Benmansour, and Hanafi [27] developed an ant colony optimisation (ACO) for the preventive maintenance scheduling problem. They tested it with a two-year horizon and approximately thirty projects. Using a commercial solver (CPLEX MIP), they solved 62% of instances within three hours. Using the ACO algorithm, they reached the optimum in 55% of these cases, with an average runtime of 200 seconds for ACO and 4808 seconds for the commercial solver.

Luan, Miao, Meng, *et al.* [28] worked on combined optimisation of the train and maintenance slot schedule. Numerical results were presented that demonstrate the benefits of simultaneous scheduling and time tabling, as compared to a sequential method.

Lidén and Joborn [29] present a mixed integer programming model for solving railway traffic and network maintenance simultaneously. A long term tactical plan is created, that aims to reduce the train operating cost and maintenance cost. Train operating cost is measured by total train running time, cancellations, and deviation from preferred departure. Maintenance cost is measured by the direct work time and the overhead time. The model is demonstrated on a weekly problem with synthetic test instances. Lidén [30] extends this model with crew resource considerations. Lidén [31] continues on this by doing a case study on a single-track railway line. In this case study, planning train traffic and maintenance in an integrated manner improves upon sequential scheduling, with 11-17% maintenance cost savings without incurring a large train traffic cost increase.

Zhang, Gao, Yang, *et al.* [32] used an enhanced genetic algorithm (GA) for solving a maintenance scheduling problem, which considers the ‘importance’ of a track segment. It also combined maintenance and train scheduling, specifically for night maintenance and sunset-departure and sunrise-arrival trains.

D’Ariano, Meng, Centulio, *et al.* [33] created a bi-objective optimisation problem for minimizing the deviation from the train planning, and maximise the number of maintenance works. It included robustness to stochastic disturbances to the train travel time and maintenance work.

Zhang, Gao, Yang, *et al.* [34] introduced a heuristic algorithm using Lagrangian relaxation for solving the timetable and maintenance problem. This included a dynamic constraint-generation technique in the iterations of the sub-gradient optimization procedure. They apply the algorithm to a practical problem concerning the Chinese railway network.

Furthermore, some novel aspects in the research done since this review in 2020 will be highlighted. Most work found since 2020 was however not closely related to this thesis, because it focused on condition-based maintenance [35]–[41]. This thesis instead uses a predetermined set of maintenance projects.

Kalinowski, Matthews, and Waterer [42] proposed a model for the annual maintenance schedule of a large railway network in Australia. It uses a granularity of one hour, and a matheuristic is designed that allows solving of the full year problem. The objective is to minimise the total capacity reduction.

Zhang, Lusby, Shang, *et al.* [43] suggested a new model, which includes platforming besides train timetabling and maintenance scheduling. This model created a daily schedule.

Mohammadi and He [37] used a deep reinforcement learning algorithm, for optimising a maintenance policy. The reward is defined as a combination of maintenance cost-effectiveness and the safety.

Bababeik, Farjadamin, Khademi, *et al.* [44] scheduled trains and maintenance on a single railway track in Iran. It uses a stochastic maintenance duration and handles different rail speed limits before and after maintenance.

2.3 Research gap

Based on the existing work, a research gap was identified. The problem handled in this thesis is different from most related work on a few aspects. Related problems from the literature often have a smaller search space. Either because the time window is smaller, meaning the created schedules were for less than a year, or because the granularity is lower. Work for example needed to be assigned a week to be scheduled in, rather than a specific starting hour. Besides, the objective is almost always less complex than the problem definition from the Dutch railways scheduling. Most problems are modelled in a linear programming model, whereas the problem definition in this work is non-linear. Specifically, the precise definition of passenger hinder with fully dynamic shortest path calculations is unmatched in any other work. The sheer number of constraints defined is also larger here than in the related work. Finally, the required maintenance is defined beforehand in this work, which is not the case for most other works.

These differences show a potential gap in research. Analysing this problem, and the differences with the earlier version from 2019, provides knowledge of how the increased realism in the objective function impacts the solving methods. This could be useful to determine how to improve other simplified problem definitions. This research will show how solving methods created on an older, simpler version still function on the more complex problem. In this work, we also propose some novel methods for improving the results, that have not been applied to the railway maintenance possession scheduling problem before.

3
Detailed Problem Description

In this chapter, the scheduling problem as defined by the Dutch railway manager is explained in depth. The objective of this problem is to find a good annual maintenance schedule. The problem input will first be defined in section 3.1. A complete schedule assigns a starting date to every maintenance project. To evaluate a schedule, an objective consisting of constraints and a cost function is given, which will be explained in section 3.2. Minimising the number of broken constraints is considered the highest priority. When the number of violations is equal, the lowest cost solution is preferred. The specific problem inputs used in this work and their most important characteristics are reported in section 3.3.

3.1 Input

The most important input for the scheduling algorithm is a list of maintenance work and a set of configuration variables. For the configuration, default values will always be used. Besides that, there are a few other input collections, which specify more static parts of the problem. These will also be the same in all experiments done for this thesis.

3.1.1 Maintenance projects

The first important part of the input is a collection of maintenance work that needs to be scheduled. For this, ProRail uses some specific terminology. For the maintenance that needs to be done, a set of projects is created by ProRail. The project management can then make requests (also called *project requests*) for work on that project. The terms request, project request, and sometimes block will be used interchangeably, signifying a unit of work that needs to be scheduled. The annual schedule is only used for project requests that hinder passengers or goods. As such, each project request requires a track possession, meaning that a part of the railway network is taken out for a certain period to perform one unit of maintenance work. Each project request needs to be assigned a single starting date; it always needs to be scheduled in entirety, meaning it can not be split or paused.

Project requests can be clustered if work needs to be done simultaneously. All requests that are a part of the same cluster need to be scheduled within the period of the longest request in the cluster. There are some other constraints or exceptions relating to the clusters, which will all be explained in section 3.2.1. Requests which are a part of the same project do not necessarily have to be planned together.

There are two types of requests; regular project requests and concept project requests. For the most part, these are the same. The main difference is the level at which the hinder is defined. Regular project requests specify exactly which railway tracks are hindered. Concept project requests instead define a larger part of the network; the minimum parts where hinder can be computed, with percentages specifying how much of the freight and how much of the passenger travel is blocked. This larger part is called a trajectory part, these terms are further explained in section 3.1.3. For the planning input this thesis works with, only concept project requests were used, for uniformity. Since the handling of concept and regular requests is similar, it is not expected to have a large influence on the results. All requests mentioned in this report are concepts, so this will not be explicitly specified from here on.

Each project request is specified with a length (in hours), an earliest required start time and a latest required end time. Besides that, the corresponding construction costs and personnel costs are defined. These are scaled based on the assigned planned period. The other costs are constant extra costs, incurred independent of the start time. For each project request, the required amount of essential specialised personnel for three types of personnel with limited availability is specified. These types are overhead contact line (OCL) personnel (Dutch: 'bovenleiding', abbreviated as BVL), exothermic welding (ETW) personnel (Dutch: 'exothermisch lassen', abbreviated as THL), and 3) commissioning (CMS) personnel (Dutch: 'Bedrijfsklaar maken, functietesten and indienststellen', abbreviated as BFI). All parts of the railway network that are hindered by this request are also specified, with a factor of hinder for goods and passengers.

Besides these requests, there is also a collection of pre-planned hinder specified. These are track possessions that already have a fixed period.

3.1.2 Configuration input

Some basic configuration input also has to be defined. This includes some simple configurations, such as the time window within which all projects need to be scheduled. Besides, it specifies some details for how the schedule should be evaluated. This includes, for each of the constraint types, whether it is considered 'hard' or 'soft' and what the corresponding penalty is. These inputs and their default values are further explained in section 3.2.1. The configuration also includes the weights for certain parts of the objective function, as well as some extra global limits that a schedule should adhere to. The exact meaning of these configuration input values and the used default values are all specified in section 3.2.

3.1.3 Other input

Besides the project requests and the configuration, there are a few other problem aspects that need to be specified to create a maintenance schedule.

Firstly, all track specifications are given. Every piece of railway track, station, bridge, etcetera is specified. For these specifications, a few definitions are required:

- **Scheduling point.** A scheduling point is a point where the railway parts are split up. There are multiple types of scheduling points, such as stations, bridges, or junctions. Switches are not included in the used network, so there is no detail on where you can go from one parallel track to another. The layout within stations is also not specified.
- **Macro track.** A macro track is an independent piece of rail between two scheduling points. This track can be used in both directions.
- **Trajectory part.** A trajectory part is one level higher in abstraction, it represents a minimal unit that can be hindered. It has a link to each of the possible macro tracks that can be used. To allow trains on this trajectory part, it needs to be supported by at least one macro track. Every macro track can only be used by one trajectory part at a time.
- **Subcorridor.** A subcorridor is a collection of one or more trajectory parts.
- **Transport flow.** A transport flow represents a flow of trains. This is a collection of multiple subcorridors. A subcorridor can be part of multiple transport flows.
- **Corridor.** A corridor is a collection of one or more transport flows.

All these collections need to be specified for the full Dutch railway network. For visualisation purposes, the coordinates of the corresponding locations are also given for each scheduling point.

In order to determine the impact of the maintenance on passengers, all information on passenger travel is specified: a collection of origin-destination pairs of passengers with the amount of passengers that are travelling this route at a reference hour. These are the regular passenger streams. Every pair with less than one per hour is filtered out. For every hour within the input period, a factor is given that specifies how busy that hour is in comparison with the reference. For example, in the middle of the night, there are very few passengers, so the factor is low. During peak travel hours, this factor is higher. Besides, there is a collection of extra passenger streams. These passenger streams represent travel that is not happening throughout the year equally, but is only present at specific dates or times. For instance, passengers travelling to certain events or tourist spots. These streams are therefore specified for a set of specific hours and a specific number of passengers at that hour. For every stream, regular or extra, a default route is given. This is the route that passengers are expected to take when they are not hindered. This route is given as an ordered list of stations where their train stops. Besides, all holiday information is given. Every holiday has a name, start time, end time, region, and factor. Any passenger stream starting in or ending in this region then has its passenger amount multiplied with this factor during the holiday.

The input also contains a good deal of information on freight trains for determining the impact on freight travel. A collection of freight streams, specified with a number of trains per hour, start date, end date, route, and blockage information, is provided as input. This blockage information specifies what happens to the freight when the default route is being blocked. This is specified with a percentage that will be rescheduled, a percentage that will be cancelled, and a percentage that will be detoured. For rescheduling and cancellation, a penalty is defined. For detouring, one or more alternative routes are given with a percentage and the extra travel hours for this alternative route.

The constraints are another important part of the input. Lastly, there are some small other inputs related to the objective, such as the personnel cost scale factors for every hour in the input period and the amount of essential specialised personnel that is available. These small constraint and cost inputs are further explained in section 3.2.

3.2 Objective

The objective consists of a set of constraints and a cost function. Minimizing the hard constraint violations is most important. In case of equal number of broken constraints, the lowest cost solution is preferred. However, the final schedule is determined by a team of schedulers. Their final wishes can impossibly be captured perfectly in a cost function and a set of constraints. Therefore, being able to generate multiple possible schedules is also important.

3.2.1 Constraints

An extensive set of constraints is defined. Ideally, a maintenance schedule should satisfy all these constraints, but this is generally impossible. Therefore, there is a prioritisation added to the constraint types. Some constraints are considered hard. Satisfying these constraints is the first priority. Other constraints are configured as soft constraints. Breaking these constraints results in a penalty in the cost function. The division between which constraints are considered hard or soft can be configured differently for each run. Constraint types can also be excluded for a certain run.

Every constraint type also specifies how violations are counted. Some constraints can only be broken once, such as a request being scheduled too early. Others can result in a larger number of violations. For example, if too much personnel is required, the number of violations is related to the amount of personnel exceeding the global limit.

In this section, each of the constraints is explained. All constraints are also listed, with their default severity configuration, in table 3.1.

Required time span. For each of the project requests, a time span is given within which the project request should be planned. If the request is planned outside this time span, this is considered one violation. In the configuration, a distinction is made between starting too early or ending too late. By default, ending too late is considered a soft constraint violation, whereas starting too early is a hard constraint violation.

Clusters. Some requests are defined as a cluster. These requests should all be scheduled together; all shorter requests of the cluster need to be scheduled within the period of the longest request. For each request in the cluster that is not completely contained in the longest request, there is one violation. Besides that, there is a violation for every request that is not a part of this cluster, but overlaps on the same trajectory part.

Personnel. There is only a limited amount of specialised personnel available for the maintenance. There are three types of this personnel. By default, there is 70 CMS personnel, 100 OCL personnel and 100 ETW personnel available. Every project request specifies an amount of required personnel for each of these types. If, on a certain day, the maximum amount of required personnel exceeds the available amount, this results in a broken constraint. The size of the violation is the amount of excess required personnel on this day. So if, for example, request A is scheduled from Monday until Friday morning and requires 50 CMS personnel. Request B requires 25 CMS personnel and is scheduled from Thursday afternoon until Sunday. Then, there is a violation of size 5 on both Thursday and Friday, so this schedule has 10 constraint violations for the constraint type 'Essential CMS personnel'.

Location conflicts. The problem definition also contains a large conflict matrix. There are certain pairs of trajectory parts which are not allowed to be blocked simultaneously. Each of these pairs also has a type, based on the reason of the conflict. There are five types of conflicts: border, corridor, junction, goods detour, and train maintenance facility. One violation is counted for each pair of project requests with at least one conflicting pair of trajectory parts between them. These conflicts can also exist between a planned request and a pre-planned hinder. There are three exceptions for the conflicts. First, if there is one request which blocks both of the trajectory parts, these two can not cause a conflict during this request. Second, some conflict types are only considered if there are passenger or freight trains at that time. Third, if a request is a concept request with a hinder lower than a configurable percentage (default 30%), this request can not be a part of a location conflict.

Weekdays. Two constraints are defined relating to the division of work between the week and the weekend. The first constraint is the minimum percentage of hours that need to be planned on weekdays outside the holiday period. The threshold can be configured and the default value for this is 25%. If too little work is scheduled during the week, the size of the violation is equal to the number of hours below the threshold. This required weekday maintenance work should be spread over different locations. The second constraint is therefore a maximum number of weekdays that can be blocked per trajectory part. This maximum number of weekdays is set for each trajectory part separately. Any day with at least one hour of project request counts towards this maximum. One violation is counted per trajectory part where the number of days exceeds this threshold. The current default values are larger than the amount of weekdays in a year. Therefore, this constraint is not influencing the objective in this thesis.

Dependencies. Dependency constraints specify a period during which certain trajectory parts can not be hindered. There are multiple types of dependencies:

- *Foreign dependencies*. Dependencies determined based on the maintenance calendar in neighbouring countries.
- *Road dependencies*. Dependencies given by the institute responsible for construction and maintenance of waterways and roads (Rijkswaterstaat). When, for example, large road maintenance is done, the alternative trains should not be hindered at the same time.
- *Event dependencies*. Some large events constitute a dependency. Events are prioritised on five different levels, where dependencies from Events-1 are the most important and dependencies from Events-5 are least important.
- *Unwritten rules*. Some dependencies are categorised as unwritten rules. For example, during some work in the Port of Rotterdam, certain trajectory parts should not be hindered.

Prerequisites. Project requests can have a prerequisite project request defined. If a prerequisite project request is not scheduled earlier, this constitutes a violation.

Constraint category	Optional Sub-Type	Severity	Penalty for violating soft constraint $(\cdot 10^{-3})$
Conflicts	Border	Hard	0
	Corridor	Hard	0
	Goods detour	Hard	0
	Junction	Hard	0
	Passenger detour	Hard	0
	Train maintenance facility	Hard	0
Train free period	Max per subcorridor	Warning	0
	Max per transport flow	Warning	0
	Max weekends transport flow	Warning	0
	Min time between subcorridor	Soft	9
	Min time between transport flow	Soft	9
Dependencies	Foreign	Hard	0
	Road	Hard	0
	Unwritten rule	Warning	0
	Events-1	Hard	0
	Events-2	Soft	47
	Events-3	Soft	28
	Events-4	Soft	9
	Events-5	Warning	0
Concurrency	Max requests at 1 subcorridor	Hard	0
	Max concurrent globally	Hard	0
	Max concurrent per traj. part	Hard	0
Personnel	Commissioning (CMS)	Hard	0
	Overhead contact line (OCL)	Hard	0
	Exothermic welding (ETW)	Hard	0
Required timespan	Request starting before	Hard	0
	Request ending after	Soft	9
Cluster	Request planned outside	Hard	0
	Planned overlapping	Hard	0
	Standalone request overlaps	Hard	0
Prerequisite		Hard	0
Too much work in a quarter		Exclude	0
Individual request planned overlapping		Hard	0
Non-individual request planned individually		Hard	0
Max weekdays per trajectory part		Hard	0
Min percentage planned during weekdays		Exclude	0
Min days between post change		Warning	0

Table 3.1: All constraints defined for a maintenance schedule with their default severity configuration. A hard severity means that satisfying these constraints is most important. Breaking soft constraints results in a penalty. This penalty is also specified in the table. A warning severity means that no cost is added, but a warning is shown to the user. The severity level exclude means nothing is done about this constraint type. For each constraint, it is also possible to specify an aggregation method. The default aggregation for each of these constraints is linear, which means that it simply uses the size of violation. For soft constraints, the corresponding penalty is multiplied with this size. It is also possible to use a constant or exponential aggregation.

Train free periods. A few constraints are defined on the number of train free periods. Train free periods are periods where there is at least twenty-four hours of hindering maintenance work. Hindering maintenance work is defined as maintenance with a passenger hinder percentage of at least 30%. The following constraints are defined:

- Maximum number of weekends with a train free period per transport flow, default 13
- Maximum number of weekends with a train free period on a subcorridor, default 6
- Minimal days between two train free periods per transport flow, default 9
- Minimal days between two train free periods on a subcorridor, default 25

Concurrent work. Per subcorridor, only a certain number of project requests are allowed simultaneously. The default configured value for this is five. This constraint is not broken if all project requests are a part of the same cluster. Besides that, there is also a maximum amount of project requests from the same overarching project allowed at the same time. By default, there are at most two requests longer than eight hours allowed simultaneously, and at most ten requests of shorter length. There is also only one request per overarching project per trajectory part allowed at any time.

Too much work in a quarter. Preferably, the work is split evenly over the four fiscal quarters. For this, a constraint violation is counted for each hour differing from the perfect split. This constraint can therefore almost never be satisfied. It was specifically designed to be used as a warning or soft constraint, and it is by default excluded.

(Non-)Individual requests. Some requests are defined to be individual or non-individual. Individual requests should not be scheduled to overlap other requests on the same trajectory part. Any request that overlaps with an individual request counts as one violation. Non-individual requests contrarily have to be overlapping with another project request at the same location during each hour they are planned at. This is used for simple and unimportant work which does not justify hindering any trains by itself, such as painting.

Minimal days between post change. Some project requests have a post change defined. A post change means that the project changes railway traffic post. There must be at least 28 days between two of these post changes using the default configuration.

3.2.2 Cost function

The used objective function for a maintenance schedule is a minimisation objective, also called a cost function. These two terms are used interchangeably throughout this thesis. The lowest number of broken hard constraints makes for the best solution, where the cost is used for comparison when the number of violations is equal. The objective is an aggregation of a few separate, conflicting parts. It could therefore also be defined as a multi-objective problem. This alternative formulation was researched in Oudshoorn [5], but was not further used due to significantly worse results. Therefore, this work will only consider the single-objective formulation.

The cost function z is determined from three parts: the financial costs (c_{fin}), the hinder costs (c_{hinder}), and the costs from soft constraint violations (c_{pen}). The cost of a schedule is defined as follows, where w_x is the weight factor for part x. The default value is 1 for each of these weight factors.

$$z = w_{fin} \cdot c_{fin} + w_{hinder} \cdot c_{hinder} + w_{pen} \cdot c_{pen}$$

Financial costs. The financial costs consists of constant other costs (c_{const}) and personnel costs ($c_{personnel}$). These costs are defined as the sum of the costs of each scheduled project request:

$$c_{fin} = \sum_{p \in P} c_{personnel_p} + c_{const_p}$$

The personnel costs are multiplied based on the predefined personnel factor at the scheduled hour h (w_h). For example, personnel is more expensive at night hours or in weekend hours. The personnel cost for a request p planned on time period T_p is therefore:

$$c_{personnel_p} = \sum_{h \in T_p} w_h \cdot c_{scaled\_personnel_p}$$

The personnel costs also have to be scaled to a minimum shift length of eight hours. If a project request p has a length L_p shorter than eight hours, and there is no request planned directly adjacent on the same trajectory parts, the cost needs to be scaled to eight hours.

$$c_{scaled\_personnel_p} = \begin{cases} c_{raw\_personnel_p} / L_p \cdot 8 & \text{if } L_p < 8 \text{ and no adjacent request on traj. part}, \\ c_{raw\_personnel_i} & \text{else}. \end{cases}$$

Hindrance cost. The hindrance cost is split into costs related to hindering passenger trains and costs related to hindering freight trains.

$$c_{hinder} = c_{passenger} + c_{freight}$$

The passenger cost is partly made up of a penalty for extra travel time required for passengers, which is computed for all hindered passenger streams S_{pass}. For each passenger stream that is hindered, it is assumed that passengers take the fastest available alternative route. The cost is the fraction of passengers blocked b_s times the number of passengers n_s times the extra travel minutes for the shortest alternative path ETM_s (Dutch: 'extra reizigers minuten', abbreviated as ERM). This delay in minutes is multiplied with a pre-defined penalty w_{ETM} . Besides that, there is a cost for providing replacement buses.

$$c_{passenger} = c_{bus} + w_{ETM} \sum_{s \in S_{pass}} b_s \cdot n_s \cdot ETM_s$$

This cost is computed based on the number of passengers requiring a bus on their shortest available detour route, for every subcorridor they use the bus on. The set of all subcorridors is defined as C. The amount of passengers from stream s requiring a bus on subcorridor c is defined as $p_{s,c}$.

$$c_{bus} = \sum_{c \in C} BusCost(\sum_{s \in S_{pass}} p_{s,c})$$

The bus cost $BusCost$ for n passengers requiring a bus is defined using a table of costs for transporting passengers on that subcorridor by bus. The lowest value from the table is used, for at least as much passengers as the number of people requiring alternative transport. So, for example, if 675 people require alternative transport, the nearest row in the table is for 700 passengers. In general, the values in this table are computed as a small constant plus a set cost per passenger.

The freight hinder cost is the sum of costs for all freight travel streams which are hindered ($S_{freight}$). For every hindered stream f, one predefined part of the freight trains is cancelled (p_{f_canc}), one part is rescheduled (p_{f_resch}), and one part is detoured (p_{f_det}). For every cancelled train and every rescheduled train, a fixed penalty is given; pen_{canc} and pen_{resch} respectively.

$$c_{freight} = \begin{cases} \sum_{f \in S_{freight}} p_{f\_canc} \cdot pen_{canc} + p_{f\_resch} \cdot pen_{resch} + c_{det_s} & \text{if } |R\_alt\_av_s| > 0, \\ \sum_{s \in S_{freight}} p_{f\_canc\_nodet} \cdot pen_{canc} + p_{f\_resch\_nodet} \cdot pen_{resch} & \text{else}. \end{cases}$$

If there are available detour paths, the detour costs of stream s: $c_{det,s}$ is computed based on the delay in hours. This delay is also called the extra freight hours: EFH (Dutch: 'extra goederen uren', abbreviated as EGU). This delay is multiplied with a cost factor defined for the freight: c_{EFH}. To determine the extra travel time, a fixed list of alternative routes R_alt_s is given for every goods stream s. For each route, the desired percentage of detour trains p_{s_r} using this route r is defined. The desired split is scaled to only the non-blocked alternative routes $R\_alt\_av_s \subseteq R\_alt_s$.

$$c_{det_s} = p_{g\_det} \cdot C_{EFH} \sum_{a \in R\_alt\_av_s} EFH_a \cdot (\frac{f_a}{\sum_{a \in R\_alt\_av_s} f_a})$$

If there are no available alternative routes $R\_alt\_av_s$, the detoured part of the freight trains are rescheduled and cancelled in the same ratio as the rest of the trains.

$$p_{f\_canc\_nodet} = p_{f\_canc} + p_{f\_det} \frac{p_{f\_canc}}{p_{f\_canc} + p_{f\_resch}}$$
$$p_{f\_resch\_nodet} = p_{f\_resch} + p_{f\_det} \frac{p_{f\_resch}}{p_{f\_canc} + p_{f\_resch}}$$

Soft constraint penalties. The third and final part of the cost is the penalties from constraint violations of constraint types with a 'soft' severity. For each of these violations the penalty is given in the configuration input. So, this cost from a soft constraint c is simply the penalty (pen_c) multiplied with the amount of violations (v_c). There are different aggregation settings possible, but this aggregation is used by default for each of the soft constraints.

$$c_{pen} = \sum_{c \in C_{soft}} pen_c \cdot v_c$$

3.3 Input set characteristics

For experimentation with new solution methods, the project requests in the year 2024 are used. In chapter 8, the years 2023 and 2025 are added to study generalisation of the developed methods. For 2023 and 2024, the set of project requests was gathered in March 2023. At this time, the set of project requests for 2024 was actively being used, and is therefore considered complete and realistic. The input from 2023 was no longer as complete. For 2025, the set of project requests was gathered in August 2023, at which time not all work was fully known. Some properties of these collections of project requests can be found in table 3.2. For the most important properties, the distribution of the values is also visualised in fig. 3.1. Some experiments required a smaller problem, so these were created from the full year problems. The process of creating these as well as the most important characteristics are shortly explained in section 3.3.3.

3.3.1 2024

In preprocessing the 2024 input data set, one project request had to be filtered for wrongly modelling the required essential personnel. Personnel that is only required for part of the request was present. Since the current problem definition could not yet handle that, this personnel was considered to be required for the full length of this long request. Therefore, it was chosen to remove this project request.

An important characteristic of the 2024 input data set is its size; the resulting schedule will be very full. There are 729 project requests, with an average length of 83.6 hours. This means that almost seven hours of maintenance work needs to be scheduled per available hour.

Besides that, it can be seen that most attributes have some outliers. In the distribution of the length, it can be seen that there are some outliers of extremely long requests. These extremely long project requests are not very hindering. However, there are some longer project requests that are also hindering passengers or requiring essential personnel.

		2023	2024	2025
Project requests	total	539	729	597
	in cluster	0	11	117
	with personnel	463	414	226
	blocking	353	682	540
Length (hours)	mean	93.6	83.6	70.7
	min	12	24	24
	Q1	50.0	48.0	48.0
	median	52.0	48.0	48.0
	Q3	52.0	48.0	48.0
	max	6385	6480	1224
Blocked trajectory parts	mean	1.9	2.5	2.3
	min	0	0	0
	Q1	0.0	1.0	1.0
	median	1.0	2.0	2.0
	Q3	3.0	3.0	3.0
	max	29	23	18
Essential personnel	mean	7.5	8.0	4.8
	min	0.0	0	0
	Q1	0.0	0.0	0.0
	median	5.0	0.0	3.0
	Q3	10.0	10.0	6.0
	max	92.0	120	75
Time window size (days)	mean	283.6	265.7	277.1
	min	2	9	15
	Q1	245.5	223.0	229.0
	median	321.0	284.0	349.0
	Q3	365.0	344.0	349.0
	max	365	344	349
Possible conflicts	mean	49.1	119.4	69.5
	min	0	0	0
	Q1	0.0	25.0	2.0
	median	4.0	97.0	46.0
	Q3	91.0	193.5	118.0
	max	261	352	271
Trajectory parts	total	344	344	344
	blocked	239	200	197
Requests blocking trajectory part	mean	2.4	7.5	5.5
	min	0	0	0
	Q1	0	1	1
	median	1	5	3
	Q3	3	11	8
	max	64	40	22

Table 3.2: The most important characteristics of the maintenance work in the full year input sets. A request with personnel means that a non-zero amount of personnel costs is specified for that project request. A blocking project request blocks at least one trajectory part. The blocked trajectory parts are the number of blocked trajectory part by one request. The essential personnel is the sum of the three types of essential personnel. The time window size is the rounded number of days that is available within the required start time and required end time of the request. The possible conflicts are the sum of dependencies, pre-planned hinder, and other requests with which a project request conflicts. The requests blocking trajectory part is the number of requests which block a certain trajectory part, these values are therefore a distribution over all trajectory parts.

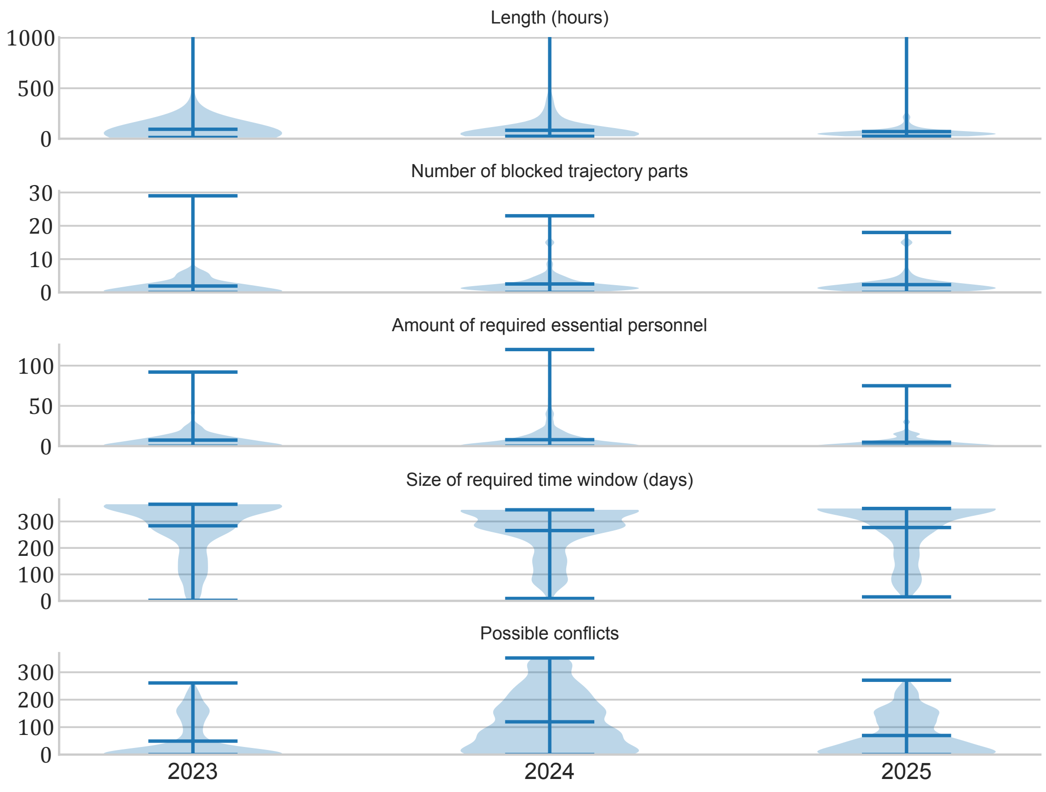


Figure 3.1: Violin plot of the distribution of the most important attributes for the full year input sets.

Most of the project requests have quite large time windows. Thus, these time windows are not expected to make the scheduling more difficult. A few outliers have very small time windows, which do seriously restrict scheduling freedom. These time windows are mostly problematic when it causes a hard constraint violation of starting too early. Starting too late is less problematic, as this is a soft constraint. In the input data set for 2024, this soft constraint is not very relevant, since the time windows are mostly restrictive in the earliest start time. The required end time is always close to the end of the year.

Another relevant property is the required essential personnel. Some requests for the year 2024 have required personnel equivalent to the limit of that type of personnel. These requests can quickly cause problems, since no other project with even a small essential personnel requirement can be scheduled overlapping.

The number of possible conflicts is also quite high, mostly with conflicts between different requests. This distribution contains fewer outliers, but some requests are conflicting with almost half the project requests that need to be scheduled. This makes it very difficult to correctly schedule these.

3.3.2 Different years: 2023 and 2025

To investigate the generalisation to different years, the project requests defined for 2023 and 2025 are used. Relevant aspects of how these were obtained will be explained. The characteristics of these years are also presented in table 3.2 and fig. 3.1. The most notable differences with the 2024 input data set are also explained.

3.3.2.1 2023

When the input data set for 2023 was obtained, not all requests were still available. These projects were mostly defined with regular project requests, but were converted to concept project requests for this thesis. The reason for this conversion is that there are some differences in the handling of concept project requests and regular project requests. Throughout the scheduling process, concept project requests are used more, so good results on these are more important. Besides, this makes it easier to compare to the results and properties of the 2024 input data set.

In this conversion, very little needs to be changed. The most important change is that the blocked tracks in the regular project requests were converted to the trajectory parts they are a part of. This conversion should generally not alter the properties too much. One request was problematic after conversion. This project blocked a trajectory part in Amsterdam for many weeks. This was not a realistic hinder from a maintenance work, so the length of this project was manually shortened.

There are a lot less requests in general, and a smaller percentage of these is hindering passengers or freight. The mean length of the requests is larger than the requests for the year 2024. The number of possible conflicts per year is a lot lower than 2024. More trajectory parts are blocked at least once by the requests for 2023, than for 2024, showing that the maintenance is more spread out over different locations.

Due to this conversion, no requests in the 2023 problem belong to a cluster. Most likely, some of them were a cluster during the original schedule creation. This makes satisfying the constraints more difficult, since being a part of the same cluster gives some constraint exceptions, most importantly for the ‘concurrent work per overarching project’ constraints. These are therefore expected to be more difficult to satisfy for the maintenance schedule for 2023.

3.3.2.2 2025

There were 597 project requests specified for the schedule of 2025. This set of projects is not expected to be complete yet. Besides, the requests are less precisely defined. This can be seen in the lower number of requests with specified personnel costs. The mean amount of required essential personnel is also lower than it was for the requests of the year 2024.

A lot more clusters are specified for the year of 2025. More than 100 requests are a part of a cluster. There is no clear reason for this large increase in the amount of clusters. The schedulers are possibly appreciating cluster definitions more now, when new requests are made. The expected consequence of this is that, contrarily to 2023, it will likely be easier to schedule without constraints. This input year also has less, on average shorter, requests with less possible conflicts. Therefore, as might be expected from a less complete problem definition, this seems like an easier scheduling problem.

3.3.3 Sub-problems

For multiple experiments, a smaller, faster to solve scheduling problem is used, when many replications are necessary. A method to create these was therefore designed. This uses a good existing candidate schedule, which is split in either six 2-month periods or four 3-month periods (quarters). The reason to split an existing schedule is that it is necessary to reduce both the schedule length and the number of requests. This way, the resulting sub-problem will create a schedule that is equally full, and have a comparable part of the maintenance that has to be performed at the same time as another request is scheduled. Also, reducing the length of the schedule decreases the required runtime more than if only the number of requests would be reduced.

Specifically, each request was assigned to a sub-problem based on the period in which it was scheduled. So if the schedule is split into quarters, all requests in January, February, and March are added to the first sub-problem. Project requests that are in two periods are put in the sub-problem it is in for the largest part. For example, if a request is scheduled from March 22nd until April 2nd, it is added to the first sub-problem. For some very long requests, it is added to both. In both sub-problems, the length is reduced to the amount that was scheduled in that period. For example, a request scheduled from March 15th until April 12th, would be added with a length of sixteen days to the first sub-problem and with a length of eleven days to the second sub-problem.

For some experiments in chapter 6, a smaller set of 2024 was required, to allow a more complete hyperparameter analysis of an expensive, novel technique. For this, it was chosen to use a 3-month period. The last quarter was most comparable to the full problem, so this was used and will further be called 2024-Q4. For chapter 7, multiple sub-problems were required, for optimisation of an order function. Besides that, more evaluations were necessary there. Therefore, six two-month periods were created. These are called 2024-P1, 2024-P2, etc. In chapter 8, both the 2023 and 2025 data sets are split into quarters, since these collections contain less maintenance work. The properties of all these created sub-problems are expanded further in appendix A.

4 Current Algorithms and Initial Improvements

Understanding what makes this problem complex and how this is currently being handled is vital for improved solving. In this chapter, the first research question “How do different aspects of the problem and the currently used algorithms influence the runtime and solution quality?” is answered. This is done by first explaining the current method used to generate suitable schedules in section 4.2. The objective evaluation has to be done many times, with every solving method; the most important parts of this evaluation will be explained in section 4.3. In section 4.4, the general algorithmic improvements that were applied to both this current solving algorithm and the objective evaluation are explained. Finally, in section 4.5, the results will be used to formulate an answer to the research question.

4.1 Background

Greedy heuristics are simple and powerful techniques, and basic greedy algorithms are generally easy to design and implement. For a more detailed explanation of greedy algorithms and its applications, we refer to the textbook by Jon Kleinberg and Eva Tardos [45].

The basic principle behind greedy algorithms is to construct a solution by repeatedly adding a solution component to the partial solution. The current addition with the highest benefit is always selected. The benefit of an addition can be calculated either by applying the regular objective evaluation to the partial solution or by using a custom evaluation. This results in relatively fast solutions. For some polynomial time problems, a greedy algorithm can find a guaranteed optimal solution, such as the Kruskal algorithm for minimum spanning trees. There are also NP-hard problems where a proven approximation bound exists for a greedy algorithm [45].

There are many problems to which greedy algorithms have been effectively and efficiently applied, such as conference paper assignment, cube packing and scheduling [46]–[49]. In the basic version, this algorithm is deterministic; it finds a single solution. Greedy algorithms therefore require a smaller amount of memory than population-based techniques.

A greedy algorithm has a very flexible framework, and as such, it can easily be combined with other algorithms. It is commonly used for finding a good initial solution in scheduling problems [50]. Another interesting approach is the iterated greedy algorithm. This algorithm repeatedly uses this construction heuristic combined with a destruction process. Iterated greedy algorithms have in the past been used to obtain state-of-the-art results on different permutation flow shop scheduling problems [51]–[53].

4.2 Greedy constructive algorithm

The currently used method to create a yearly maintenance schedule is a greedy constructive algorithm. This algorithm first orders the maintenance work that needs to be scheduled based on a predefined function. Clustered project requests are scheduled first. These clusters contain multiple requests and thus probably affect more trajectory parts. Secondary, the requests are ordered based on the amount of hindrance the request is expected to cause passengers. Requests hindering many passengers should be scheduled earlier, since they benefit more from periods with less passenger travel, such as weekends or holidays. The time window size is used as a tiebreaker, and is therefore defined to always be smaller than one. Requests with a smaller time window are scheduled first. By default, the order function for project request x with affected trajectory parts $ATP(x)$ with a total amount of days in the schedule D is therefore as follows:

$$\begin{aligned} score(x) = {} & IsCluster(x) \cdot 10^9 \\ & + Length(x) \cdot \textstyle\sum_{t \in ATP(x)} NumPassengersPerTrajPart(t) \\ & + \frac{D - TimeWindow(x)}{D} \end{aligned} \tag{4.1}$$

After this ordering, the requests are greedily assigned a starting time one-by-one. This is done by trying out all possible starting times, with a specified granularity. By default, this means every possible starting day within the request's required time window is attempted. The time of day is determined such that the request will end at 05:00 AM. The best of all attempted starting times is chosen. This process is repeated until the schedule is complete, so all requests have an assigned start time.

Whenever the request is a part of a cluster, the full cluster will be planned. This is done by first finding the longest request(s), which is then greedily assigned a start time. After this, each of the shorter requests are greedily scheduled within the time window of the longest request.

This algorithm is relatively efficient and requires little computation time, since it never moves project requests that have already been scheduled. Besides, it does not include the requests that are still to be added to the schedule in making the current scheduling decision. Previous research has shown that some good results can be obtained using this method. The order function is very important to obtain these results.

This algorithm also has some important downsides. Because it never moves requests again, only a limited number of full schedules is tried; exploration is limited. Furthermore, the current request will be assigned the best time, independent of the preferences of other requests. This can be detrimental to the solution quality. This happens for example if the first request has multiple starting times with very similar costs, where a request which will be planned later can only be scheduled on one of those times.

For an earlier version of this problem, more strategies to find good schedules were developed [5]. Due to the updates to the objective and the resulting increase in objective evaluation time, these strategies became infeasible. More details on these strategies, and a comparative analysis of how they perform on the updated problem, will be presented in chapter 6.

4.2.1 Randomness

The greedy constructive algorithm is deterministic, as the optimal starting time is always chosen. However, for multiple reasons, it is beneficial to introduce some randomness. The first reason is that these schedules may have better final solution quality, and the best option from multiple attempts can be used. Secondly, this randomness allows better usage of this algorithm as a part of other algorithms. The heuristic can, for instance, be used to create as a diverse starting population for an evolutionary algorithm. Lastly, it makes it possible to more reliably measure the performance, which is necessary for the experiments with solution quality done in this thesis.

In the work that proposed this algorithm, some comparisons were done with multiple ways of introducing randomness [5]. The two main methods were randomisation of the assigned start time and randomisation of the order in which requests are assigned a start time.

The first method uses a list of probabilities to decide on a starting time from the best few possibilities. So, for example, with the randomisation list (0.5, 0.35, 0.15), there is a probability of 0.5 that the best starting time is selected, which is the same choice as the deterministic algorithm. With probability 0.35, the second-best option is assigned to the request and with probability 0.15, the third best starting time is assigned.

Similarly, with order randomisation, the order is randomised using a list of probabilities. In this case, the deterministic order function is first used. Whenever another request should be added to the schedule, the probability list is used to decide which of the upcoming requests are scheduled.

The comparative research showed that, for the 2019 and 2020 input years and the old version of this problem, the next request randomisation had on average a lower number of constraint violations, whereas the next starting time randomised runs had a lower average cost. Based on these results, it was chosen to use the order randomisation for all greedy algorithms with introduced randomness, as minimising constraints is considered most important. The proposed probabilities, that will also be used in this thesis, are (0.5, 0.35, 0.15).

4.3 Current objective algorithm

This objective function is computed incrementally based on changes to the schedule. Unchanged part of the objective do not have to be recomputed this way. The basic principles and most important aspects of this algorithm are explained and analysed in this section. More details concerning this incremental computation can be found in the thesis by Oudshoorn [5].

The algorithm is based on two basic operations: adding and removing a request. Moving is executed by doing a remove operation followed by an add operation. Project requests are also called blocks in this context. Throughout this process, the following things are stored and updated after every add or remove operation:

- The currently planned blocks and their starting times
- The currently blocked tracks per hour
- The current constraint violations and their penalties
- The overlapping periods
- The added blocks since the last availability cost computation
- The removed blocks since the last availability cost computation
- The current financial costs

An overlapping period is a period where at least one block is scheduled. These are used to keep track of constraints and the financial costs. This can be done on many levels, which allows to easily see when project requests with a certain property have changed. More explanation of how these overlapping periods work, which levels are used, and how they are updated is explained in section 4.3.1. Just before the objective evaluation is required by the algorithm, the availability costs are also updated. This process is explained in section 4.3.2.

4.3.1 Overlapping periods and constraint handling

Keeping track of the overlapping periods and the corresponding constraints is an important part of the scheduling, which requires a substantial part of the runtime. A scheduling run of the 2024 input year was profiled to quantify these computational demands. At the beginning of the scheduling, around of quarter of the runtime is spent on this part of the objective. When the schedule gets fuller, this part becomes larger. Towards the end, around a third of computational time is used for these computations.

Every constraint type is being tracked by a handler. These are two types of constraint handlers: general handlers and handlers based on overlapping periods. When a block is added or removed, the general handlers are notified of the operation and alter their state based on this. All overlapping periods are updated with the change. For each resulting change to an overlapping period, constraint handlers on this level are notified and update their state.

Overlapping periods. An overlapping period is a period in which one or more project requests are scheduled. It is never adjacent to or overlapping with another overlapping period of the same type. If, for example, request A is scheduled from 00:00 to 04:00 and request B is scheduled from 02:00 till 06:00, this results in a single overlapping period from 00:00 to 06:00. This overlapping period is on a global level, but many levels are possible. For example, on a trajectory part level, request A hinders trajectory parts X and Y and request B hinders trajectory part Y. In this case, trajectory part X has an overlapping period from 00:00 to 04:00 and trajectory part Y has an overlapping period from 00:00 to 06:00. The following levels are used: global, transport flow, trajectory part, subcorridor, post change and overarching project. The global level contains only one collection of overlapping periods. On the other levels, multiple collections are stored, one for each property value. So, for every trajectory part that is blocked at least once, a collection of overlapping periods is kept.

Every update to these overlapping periods is relatively simple. If a block is added, it needs to be added to every overlapping period collection it belongs to. So, it is added to the global collection of overlapping periods, but also, among others, to the overlapping period collection for every trajectory part it hinders.

Adding a request to a collection of overlapping periods is done by looking at the current overlapping periods. If it overlaps with none of them, a new overlapping period is added. If it overlaps with one of them, it is added to this one. If it overlaps with multiple existing overlapping periods, these are all merged. This is done by removing all existing periods, and then adding one new overlapping period with all their requests and the new request combined.

There is a reference stored from every request to all its overlapping periods. Then, when a request is removed, each overlapping period it was a part of is updated. If the block is the only one in the period, it is removed. If there are still blocks left, every hour of this block is checked. If there is no other project request in the overlapping period still scheduled during the hour, it is removed from the period. This may result in the overlapping period being split in multiple overlapping periods.

During these updates, all handlers that are registered to this level are updated using four types of notifications. One notification for every removed overlapping period, one notification for every request removed from an existing period, one notification for every new overlapping period, and one notification for every project added to an existing period. These updates are then used by each handler to update their state.

Financial cost updates. The financial costs are updated using a handler that is notified of any change to the global overlapping periods. When a new overlapping period is created, the personnel and construction costs for that period are calculated. This is done by first checking each request with a length shorter than eight hours. If it is not scheduled directly adjacent to another request, the personnel costs are scaled to a shift of at least 8 hours. After this, each hour of each request is scaled with the factor of personnel at that hour and added to the total cost for that overlapping period. The cost calculated for this overlapping period is then added to the overall cost, as well as stored separately. Whenever an overlapping period is removed, this stored cost can then simply be subtracted from the total cost. When a notification of a request added to or a request removed from an existing overlapping period is received, this is handled by simply removing the current cost of the existing overlapping period and then adding the cost of the changed overlapping period in the same manner as explained for a new overlapping period.

General constraint handlers. Whenever a constraint can be considered based on add or remove operations without using information from overlapping periods on any of the defined levels, this constraint is handled by a so-called general constraint handler. This handler keeps track of the current violations and its current state. The handler is notified of any addition or removal. The following constraint types are handled by a general constraint handler:

- Prerequisites
- Request planned outside cluster
- Too much work planned in a quarter
- Minimum percentage during weekdays
- Required time span

An example of such a general constraint type is the prerequisite constraint. This constraint ensures that certain requests must be scheduled after all of its prerequisite maintenance work has been finished. Whenever a request is added, its prerequisites are checked. Each prerequisite that is scheduled, but not finished before the new request means a constraint violation is added. Secondly, all requests that have the newly added request as a prerequisite are checked. In the same manner, if that other request has a scheduled starting time before the end time of the new block, a constraint violation is added. When a request is removed from the schedule, all related prerequisite constraint violations are removed.

Overlapping period constraint handlers. Some handlers use the current state of the overlapping periods to determine whether the constraint has been broken, and how many violations the current schedule would give. Each handler is defined to act based on a certain level of overlapping periods. Whenever a new request is added to the schedule, the overlapping periods are updated. Each update to these overlapping periods is then used to update the constraint status. The following constraints are handled by an overlapping period constraint handler:

Global level

- Personnel
- Conflicts

Transport flow level

- Train free periods per transport flow
- Maximum weekends per transport flow

Trajectory part level

- Cluster overlaps
- (Non-)Individual requests
- Maximum weekdays per trajectory part
- Dependencies

Subcorridor level

- Concurrent work per subcorridor
- Maximum weekends per subcorridor
- Train free periods per subcorridor

Post change level

- Minimal days between post change

Overarching project level

- Concurrent work per overarching project

These overlapping period constraint handlers are best explained using an example; the essential personnel constraint handler. These constraints are handled based on the notifications of updates to the global overlapping periods.

Whenever the essential personnel constraint handler gets a notification of a new overlapping period being added, the overlapping period is first split. Each part of the period with the same set of requests scheduled is extracted. For example, a new overlapping period consists of two project requests, A and B. Project request A is scheduled from Monday 12:00 till Thursday 12:00, request B is scheduled from Wednesday 12:00 until Friday 12:00. Then, there are three relevant parts to split the period into: Monday 12:00 – Wednesday 12:00 with only A, Wednesday 12:00 – Thursday 12:00 with both requests and Thursday 12:00 – Friday 12:00 is only B. Then, if these parts have been found, the amount of required essential personnel can easily be updated. If, for example, requests A and B both require twenty essential personnel, the following updated are done. Monday, Tuesday, and Wednesday are updated with value 20, Wednesday and Thursday are updated with value 40 and Thursday and Friday are updated with 20. When a day is a part of multiple of these unique set periods, only the highest amount of required personnel value is kept. So in this example, Wednesday and Thursday both store an amount of required personnel of 40. Whenever a day goes over the limit, the violations are stored. Thus, if the limit is 30 available personnel, there will be 2 days in violation. A reference is stored from each violation to the overlapping period it occurs in.

When the essential personnel constraint handler is notified of the removal of an overlapping period, it resets all maximum essential personnel values in the time window of this overlapping period to zero. All violations referencing this overlapping period are also removed.

When a new item is added to an existing period or an item is removed from an existing overlapping period, this is handled as first removing the period, and then adding a new overlapping period with the change.

4.3.2 Availability costs

The availability costs represent the hinder to passenger and freight trains. The computation of availability costs is responsible for more than a third of the computational time of the greedy algorithm. In the beginning of a scheduling run, around half of the runtime is spent on computing these costs. Towards the end, this decreases towards around a third of the runtime. The absolute amount of computation spent on this per added request still increases, but more slowly than other parts of the computation.

These availability costs are not immediately recomputed after every remove or add operation. Instead, it is only done just before the cost value is necessary, since this computation has a large complexity. By handling the cost changes for multiple operations at once, the required computational efforts can be reduced.

All operations that were done since the last availability cost update are used to determine during which time periods the availability costs will have to be updated. At these periods, the freight and passenger hindrance costs will then be updated separately. First, the process of determining where updates are required is explained. Then, it is explained how the passenger hindrance costs are updated in this period. Lastly, the freight hindrance updates will be explained.

4.3.2.1 Determining when the availability costs have to be updated

Updating hindrance to passengers and freight has to be done for every hour when something has changed, since each hour can have different passenger and freight travel. Therefore, the first step that needs to be done to update the availability costs is to find the hours which have been changed since the last availability costs update. The pseudocode for this algorithm is given in algorithm 4.1.

Algorithm 4.1 Update availability costs

```
1: Input RemovedBlocks, AddedBlocks: all blocks added or removed since the last availability update

2: for Removed block ∈ removed blocks do
3:     for Added block ∈ added blocks do
4:         if (Added block is same request as removed block and
            Added block is scheduled at same time as removed block) then
5:             Delete removed block from set of removed blocks
6:             Delete added block from set of added blocks
7:         end if
8:     end for
9: end for
10: for Removed block ∈ removed blocks do
11:     for Hour ∈ removed block's period do
12:         Calculate passenger hindrance costs at hour (algorithm 4.2)
13:         Calculate freight hindrance costs at hour (algorithm 4.3)
14:     end for
15: end for
16: Determine the global level overlapping periods of the added blocks
17: Group added blocks by their overlapping period
18: for Overlapping period ∈ overlapping periods from added do
19:     for Hour ∈ MinCover(added blocks in overlapping period) do
20:         Calculate passenger hindrance costs at hour (algorithm 4.2)
21:         Calculate freight hindrance costs at hour (algorithm 4.3)
22:     end for
23: end for
```

All blocks that were removed and then added again at the same time are filtered from the operation. Then all hours of the removed block operations are updated. Finally, the overlapping periods relating to the added block are found. These can be used to group added requests which may be overlapping together. Then, the minimal cover per group is determined. This is done to find all hours when one or more blocks were added. An availability cost update is done for every one of these hours.

4.3.2.2 Passenger hindrance costs

To update the passenger availability costs, the extra travel time for all hindered passengers is computed. The pseudocode of this algorithm is given in algorithm 4.2. This is done for every hour when something has changed separately. First, the stored costs of this hour are removed from the overall cost sum, and reset to zero. Then, passenger streams which are affected by the current maintenance work are found. A passenger stream is an origin-destination pair with a certain number of passengers. These streams are defined for every hour in the scheduling year. Every stream has a default route, and the trajectory parts required to use that route are known. The relation between the trajectory parts that are blocked with the passenger streams requiring at least one of those trajectory parts on their default path are cached. For all hindered passengers, the detour paths they will take with the current maintenance hinder are computed. These paths are used to compute the costs for the extra travel time and the cost for providing alternative buses.

Algorithm 4.2 Calculate passenger hindrance costs at a certain hour

```
1: Input the hour when passenger hindrance needs to be updated and the trajectory parts blocked

2: Set extra travellers minutes (ETM) at the hour to update to $0$
3: Set bus costs at the hour to update to $0$
4: if Set of passenger streams blocked by trajectory is not available in cache then
5:     Find set of streams which requires the blocked trajectory parts on their default route
6:     Add blocked streams to the cache
7: end if
8: Get blocked streams from the cache
9: Get detour paths for all blocked streams from cache or compute if not present
10: for Stream in the set of blocked streams do
11:     Compute number of passengers blocked on stream at the hour to update
12:     Increase ETM with number of passengers times difference between default and detour path
13:     if Detour path requires traveling by alternative bus then
14:         Update bus costs based on detour path and the number of blocked passengers
15:     end if
16: end for
```

To determine the amount of extra minutes the travellers have to spend on their journey, shortest path computations are required. All streams that can not use their default route due to the maintenance, are grouped based on their origin. A single source Dijkstra algorithm is used to find the shortest paths on the graph altered to include the hindrance. A single source Dijkstra algorithm means that from one source, the paths to all destinations are found using a Dijkstra algorithm [54]. So, the shortest detour path, not using any of the blocked trajectory parts, is found for every hindered passenger stream.

For efficiency reasons, this part of the objective is approximated in evaluation. This approximation is done in two ways. First, the train transfer times are not handled completely exact. This aspect of the approximation as well as more detailed explanation on these detour path computations are explained in depth in chapter 5. Second, the streams that are below a configurable threshold are left out of the computation during the scheduling algorithm. For the final schedule, these streams are included.

4.3.2.3 Freight hindrance

How the freight availability costs are modelled is explained in section 3.2.2. The pseudocode for how this is computed can be found in algorithm 4.3. The freight hindrance costs are more straightforward than the passenger availability costs. The amount of trains that will be rescheduled, cancelled and rerouted as well as the detour routes to use are predefined. For each hindered freight stream, we check which of these detour paths are not hindered by maintenance. If all detour routes are blocked, the amount of cancellation and rescheduling is increased. Otherwise, a penalty per hour delay is computed. This computation is also done for every changed hour, for every freight stream. It is less complex than the passenger hinder, but also requires a substantial computational effort.

Algorithm 4.3 Calculate freight hindrance costs at a certain hour

```
1:  Input: the hour when freight hindrance needs to be updated, and the trajectory parts blocked

2:  Set freight hindrance costs at hour to update to 0
3:  Determine freight streams that require the blocked trajectory parts on their default route
4:  for Freight stream in blocked freight streams do
5:      Determine which of the specified detour routes are available with blocked trajectory parts
6:      if At least one available detour route then
7:          Compute the ratio of detour over the available routes
8:          Determine the detour penalty based on this ratio and the amount of detoured trains
9:      else
10:         Increase the amount of cancelled and rescheduled trains for this stream
11:     end if
12:     Increase freight hindrance costs at the hour with the cancel, reschedule and detour penalties
13: end for
```

4.4 Improvements to current algorithms

During the problem analysis, some significant improvements were found to the greedy algorithm and the objective function. These improvements were not directly related to any research question or new technique. All experiments in this work were run with these initial improvements applied. At the starting point of this thesis, one greedy constructive scheduling for the year 2024 run took around twenty-four hours, with a less precise cost calculation and more constraint violations. These first improvements already decreased the runtime of a baseline greedy constructive run to only around three hours.

4.4.1 Handling clusters

In the greedy algorithm, clusters are handled by first scheduling the longest request, then each of the shorter requests are scheduled greedily within the time period of the longest request. However, this original method for handling clusters caused unnecessary hard constraint violations at times. This happened when the longest request was scheduled on a location where one of the shorter requests would not be able to be planned non-conflicting. Therefore, this has been adapted to schedule all of them together. So, the longest request is assigned a start time based on the score of adding all cluster requests in this time frame. This made the greedy solution a little slower, but it had a direct improvement in the number of constraint violations. For the 2024 schedule, the number of constraint violations reduced from an average of around 8 or 9 to an average of around 5 or 6 with this new cluster handling. Therefore, this adaptation was used for all experiments with the greedy algorithm throughout this thesis.

4.4.2 Precompute changed hours

An inefficiency was found in algorithm 4.1. Whenever the cost is required, the availability costs have to be updated. All additions and removals that were done to the schedule since the last update to the availability are stored. All availability costs are then first recomputed for the period of each of the removed blocks. For each overlapping period where one of the added requests is a part of, the hinder costs are updated. This results in unnecessary computation in two ways. First, if a request is moved, and the new time period overlaps with its old time period, the availability costs for these hours will not actually change, but will be recomputed. Second, whenever there is overlap between the periods of two removed blocks, or one removal and one addition, these hours will then be recomputed twice.

This algorithm was therefore adapted to algorithm 4.4, which resolved both of these problems. In this version, a check is done for each of the removed blocks to determine whether they are added again, and if this new period overlaps with the old period. Then, these periods are subtracted from both the removed block and the overlapping add block. This prevents the unnecessary computation from the first case. After this process, a min-cover algorithm is used to determine all hours in which a block was either added or removed. This way, an hour will not have more than one recalculation of the availability costs within one update, so the double computation from the second case is resolved. Both of these changes also have some possible downsides, that needs to be outweighed by the benefit.

Algorithm 4.4 New algorithm: Update availability costs

```
 1: Input all removed blocks and added blocks since the last availability update

 2: for Removed block ∈ removed blocks do
 3:     for Added block ∈ added blocks do
 4:         if Added block is same request as removed block then
 5:             if Added block is scheduled at same time as removed block then
 6:                 Delete removed block from set of removed blocks
 7:                 Delete added block from set of added blocks
 8:             else if Added block overlaps with removed removed block then
 9:                 Subtract removed period from added block's period
10:                 Subtract added period from removed block's period
11:             end if
12:             break
13:         end if
14:     end for
15: end for
16: for Hour ∈ MinCover(Periods of added blocks ∪ periods of removed blocks) do
17:     Calculate passenger hindrance costs at hour (algorithm 4.2)
18:     Calculate freight hindrance costs at hour (algorithm 4.3)
19: end for
```

The first check to find overlapping additions and removals requires some extra operations, especially when many blocks have been removed and added. In the original algorithm, this loop was already done to find blocks rescheduled to the exact same time. The reason this was deemed useful in the original algorithm stems from the general usage. In most situations, this recalculation of the availability costs is done after one moved block, so one removal and one addition of the same request. In this case, this check is very fast and worth it to make the situation where it is rescheduled to its current starting time faster. For example, if a move was tried and resulted in more constraint violations, it can be moved back without computing availability costs. Since these reasons are still valid and the additional check for overlap is not that much increase, the benefit of this part is expected to often outweigh the loss. Only in cases where many moves are done at once, and blocks are moved outside their current period, this will not be the case.

For the second part, computing the min cover, which requires some extra computation compared to the original algorithm. The original algorithm grouped the added blocks before computing the min cover, making it more efficient. It went over all removed blocks without a min cover algorithm. The added blocks did require a min cover, but all added blocks were grouped first. The benefit of this change is more dependent on the used algorithm. If multiple moves are made before requiring the cost calculation, the probability of there being overlap in hours between multiple operations becomes higher. However, with multiple changes, the min cover also takes longer, and the detriment from no longer grouping by overlapping period is larger. This algorithm gives most improvement when multiple moves, close together in the scheduling year, are done. It is slightly less efficient when multiple moves, spread over the scheduling period, are done. It could be even better to dynamically choose between using and not using this part based on the used algorithm and its mutations. However, in most cases, it is expected that this second part will also decrease the required runtime. Therefore, this dynamic switching was kept as future work, since it could better be added when the useful algorithms have been more extensively researched.

4.4.3 Exact objective evaluation

Currently, the objective evaluation already contains some approximation in the process of computing the passenger hindrance, for reduced objective evaluation runtime. However, when the search strategy has created a complete schedule, it is preferable to have an exact final solution. This means that different schedules can be compared based on an accurate calculation of the designed objective.

Therefore, the path computation was kept the same for evaluations during the search strategy, but an exact computation was implemented for the final schedule. The difference between this hindrance approximation and the exact computation of the extra travel time will be explained in section 5.3.

4.4.4 Memory management

In profiling results of the current implementation of the objective, it could be seen that the memory management required a large part of the computational resources. Some efficiency improvements to this aspect of the problem have been applied. There may be more to be gained here, and memory management should be considered when designing new techniques.

All algorithms are implemented in C#, which means that releasing memory is done automatically [55]. When objects are no longer being used, the memory is made available for future allocations. This is done using a generation-based memory. All new objects go to generation 0. This is the set of objects that should contain short-lived objects. Whenever garbage collection is required, this generation will first be cleaned. All objects that are not reclaimed in generation 0 are now moved to generation 1. Finally, if these objects also remain in-use when generation 1 garbage collection is done, they are moved to the final generation 2. The main advantage of this principle is that most short-lived variables can be recollected fast; objects that are long-lived will not have to be checked as often.

The incremental implementation of the objective is very memory-consuming. There are a few large caches that prevent having to recompute certain parts many times. Besides, especially when a schedule becomes full, large collections of overlapping periods on all different levels are stored, as well as different states for every constraint. Many of these variables are long-lived, and the generation 2 memory is by far the largest. Due to this large memory consumption, garbage collection becomes slower. Any time a generation 2 collection is necessary, this requires a lot of computation time, since the references of all these different objects need to be checked, and the entire generation is compacted.

Whenever the system has low physical memory available, a garbage collection will be forced. Therefore, it is important to run on a machine that can handle the memory demands of this program. Most experiments in this work are done on a machine with 16 GB of physical memory. This was enough, but for very long runs, it was necessary to implement an emptying of caches to prevent these low physical memory collections happening too often. The process also keeps an acceptable threshold by itself. Whenever memory consumption is above this threshold, collection is done. This threshold is continuously updated during the run. However, this will still cause many collections, due to the increasing nature of the memory requirements of the greedy algorithm.

Overall, it can be seen that garbage collection pressure is around 30%, meaning almost a third of computation time is spent on memory management. A few small improvements were done with limiting the amount of unnecessary large lists by specifying the required capacity. Reusing lists, rather than creating a new list for repeated computation, also gave some small improvements. The largest improvement was achieved by fixing two memory leaks. This means objects were not being collected despite them not being required any more; occupying unnecessary memory. This pressure is still high, and it is expected that more profit is achievable by focusing on a more memory-efficient implementation.

4.4.5 Improved usage of single source Dijkstra

As was explained in section 4.3.2, all required origin-destination pairs are grouped based on their origin. Then, a Dijkstra algorithm is used to find all paths from every unique origin. This is done by first labelling all nodes with the shortest found distance from the source. All edges are then traversed until the shortest path tree from the origin to any destination is created. This tree can be traversed back to generate all actual paths. The paths between the hindered origin-destination pairs are then used to compute the passenger hindrance. Often, many paths from the same origin are blocked simultaneously. As such, it is beneficial to compute these all using only a single labelling phase. This way, parts of the shortest path tree are reused and do not have to be computed multiple times.

Only the used origin-destination pairs are cached. The reason for this is that the other destination paths will often not be useful later. At most hours, the defined sets of passenger streams are very similar. So if the current hour does not contain that origin-destination pair, there is low probability that it will be required later. Therefore, the benefits of keeping the cache smaller outweighs the potential benefit of not having to recompute this tree when the other path is required later.

In the original implementation, for each possible destination, the shortest path tree was traversed back to find the complete detour route. In the improved implementation, the traversal through the search tree is done only when that origin-destination pair is actually used for computing the hinder. The labelling phase is still done for any possible destination, since checking when all required destinations have been labelled is expected to generate too much overhead for too little benefit.

4.5 Conclusion

Good understanding of what makes this problem complex is essential for improving on the currently used techniques. Determining the aspects that make finding a good schedule difficult and slow is important. The fact that it takes a long time to evaluate the quality of a full schedule is an important reason for this. It influences the runtime directly, and also makes it more difficult to find better quality schedules. To find good schedules, some exploration is required. However, search is relatively slow due to the slow objective evaluation. Speed-ups that make this evaluation more efficient have a direct positive impact on the achievable runtimes and search.

The analysis and improvements in this chapter allow us to answer the first research sub-question: “How do different aspects of the problem and the currently used algorithms influence the runtime and solution quality?” The availability costs are the most complex part of the problem definition, as these are defined with high precision. The passenger hindrance computations are most influential. Hindrance is computed separately for every hour, with fully dynamic shortest paths for every origin-destination pair. Besides, the large number of overlapping period collections that need to be updated also result in a substantial runtime, especially towards the end of a greedy scheduling run. Most constraint handlers in itself have limited impact, but they require these overlapping periods to be kept on this large amount of levels, meaning they indirectly require more runtime. The path caches and overlapping periods also require a lot of memory. The amount of physical memory in itself is not a limitation. However, management of this memory has a strong negative impact on the runtime.

4.5.1 Future work

A few interesting future work directions were identified from the results of this chapter.

From the initial profiling work, already some improvements were found. However, more improvements are definitely possible by more close analysis, especially of the most often executed code. In this chapter, the importance of spending time to optimise the most essential parts of the computation is shown. Evaluating the objective happens thousands to millions of times during an algorithm run, so even very small improvements to this can have significant impact on the usability. By removing a small inefficiency in the implementation of the shortest path finding, which is executed for all hindered passenger travel, for every new combination of blocked trajectories, a significant improvement was already be achieved.

In some cases, the most efficient algorithm for the incremental objective evaluation is dependent on the type of mutations that are done. Some ability to make the objective evaluation dynamic depending on the current situation could prove beneficial. For example, the greedy algorithm often moves a project one day forward at a time. This move is handled as first removing the project from its overlapping periods and then adding the project again to the overlapping periods. Many of the constraint handlers then also handle this removing from the overlapping period as deleting the full period and adding a new period. Then, adding the request is handled similarly as removing the overlapping period and creating a new one. More efficient computation is possible here, especially by specialising the implementations on this common usage of moving only a day.

Besides, the memory management improvements could be extended. Possibilities to store some parts of the problem status more compact could be investigated. Looking into possibilities to change the garbage collection settings to handle the large amount of longer-lived variables better could also improve the runtime impact of the memory management. A specific suggestion that might, for instance, improve memory management would be to look into a more efficient path caching method. If the path cache keeps track of which combinations of blockages are not used often, these entries could be removed. Ideally, this would be done before they reach generation 2.

Furthermore, some improvements found in this chapter emphasised a drawback of creating such a complex problem definition and solution method. If a problem is this complex, and being altered regularly, small mistakes or inefficiencies easily slip into the implementation. Especially since not every aspect of the problem definition is actively being profited from. For example, some limit values are set quite high, meaning computation to check for that limit is useless. Another example is the fact that there are no extra passenger streams defined, which are different for each hour. The possibility to have extra passenger streams had a significant impact on the algorithm design, and the knowledge that every hour has the same streams is not utilised in the computations. This is of course an extremely difficult balance, between modelling as realistically as possible, while also reducing unnecessary complexity which can cause inefficiencies. There are some model simplifications possible, which would not reduce realism that much. Moreover, allowing to configure more parts of the complexity for each run separately could improve the trade-off available for schedulers.

5 Detour Path Approximation

The passenger availability cost computation is one of the expensive parts of solving the yearly maintenance scheduling problem. An important reason for the large computational demands of this cost computation is that it has to be done for every hour when something has changed. Every origin-destination pair of passengers that is blocked requires a detour path. By applying approximation to the detour paths and the passenger streams, it is possible to speed up any search strategy, while losing some objective precision. Less accurate objective values could mean worse scheduling choices are made, so these speed-ups need to be considered in relation to the effect on the solution quality.

In this chapter, the research question “How can approximation of passenger detour paths be used for faster solving?” will be answered. First, the relevant background is given in section 5.1 The important graph characteristics are analysed in section 5.2. Then, in section 5.3, all compared strategies are motivated and explained in-depth. The experimental methods used for comparing these algorithms are explained in section 5.4, before showing the results in section 5.5. The research question will finally be answered based on these results.

5.1 Background

In this section, the required background for this chapter is explained. Some background on approximation of the objective is given. Some alternative approaches for considering passenger hinder are also given. Besides, the inspiration for some of the shortest path techniques is briefly presented.

Objective approximation. The goal of this chapter is to apply approximation to a part of the objective. Approximating the objective is a technique more commonly applied to expensive optimisation problems [56]. This is commonly used in combination with evolutionary algorithms, since the similarity to its parents can be used. This approximation can be used to more quickly search for solutions with a good fitness, and only apply the full objective evaluation for a smaller set of individuals. The approximation is sometimes also referred to as a surrogate [56]. There are many surrogate modelling techniques. The approximation can be a problem specific approximation function, or a generic model which learns the fitness. This generic model can be trained either online (during the optimisation), offline (before optimisation) or a combination of the two. Many machine learning models have been used in the literature, such as polynomial response surface, Kriging, radial basis functions and support vector machines [56].

Surrogate-assisted optimisation has been effective in numerous problem applications, and is often applied in cases where the objective evaluation is a physical simulation. Energy-efficient building, ship, satellite, or motor design are examples of problems it was successfully used for [56]. A downside of this method is that good results of the optimisation requires the surrogate to be accurate enough. This is difficult for high-dimensional problems, such as the problem used in this thesis.

A surrogate-modelling based algorithm in the field of scheduling was proposed by Hao, Liu, Lin, *et al.* [57]. This solved the bottleneck stage problem by decomposing the problem into first assigning the jobs to the different machine, and then finding the optimal sequences for each machine. Significantly better results were obtained by partly replacing this second part with a surrogate model.

Passenger hindrance computation. More maintenance problems include an approximation of the amount of hinder to passengers. Weert [8] computes the passenger hindrance based on the assumption that one of the k-best paths would be taken. In some other works, the hindrance is modelled as the amount of disruption to the train timetable [12], [26]. This indirectly also determines the passenger hindrance, but neglects the difference between busier and less busy trains. Boland, Kalinowski, Waterer, *et al.* [24] minimised passenger hindrance by maximising the total flow over time.

Shortest path techniques. In this chapter, the shortest path techniques that are used to determine the routes that will be taken by the passengers, especially when they are hindered, are considered. In literature computing the shortest paths for passengers using public transport, the routes taken by passengers are often modelled more precise, using timetable information [58]. An example of fast shortest path computation on railway graphs which uses edges independent of the time was found in Holzer, Schulz, and Wagner [59], which also applies the same technique on road networks. Shortest path calculations on road networks are often more similar, as this does not use time information. The work by Bast, Funke, Matijevic, *et al.* [60] proposes a similar solution for shortest path computation on road networks. This work uses the hierarchical structure of road networks. Longer distance paths often travel through the same important points.

5.2 Graph Analysis

In order to choose the most suitable approximation algorithms, the structure of the underlying graph was analysed. The railway infrastructure it is modelling can be seen in fig. 1.1. In this section, some basic characteristics of the graph are given. The used railway network is a weighted directed graph, with transfer times. Every node represents a station where trains can stop. Every edge represents a direct travel connection: it is possible to travel from the edge's origin to the edge's destination with a train that does not stop. In reality, whenever a passenger has to transfer from one train to another, some extra walking and waiting time is required. The average time of the full transfer from one edge to another is specified for the network. These transfer times are given for 1078 tuples of three nodes, specifying which nodes are surrounding the transfer (from, through and to). If transfer time is not specified for the edge pair, zero transfer time is counted.

First, the basic graph characteristics were computed based on the full graph. These can be found in table 5.1. The total amount of nodes and edges makes this a relatively small network. From these statistics, it can be seen that many of the edges are dominated, meaning there is another edge connecting the same nodes with a shorter edge. The most important reason for this is that there are alternative travel edges defined. These edges are meant for when a certain edge is blocked by maintenance; buses will then be provided to transport passengers. These are therefore edges on the same connection, but with a different length. A few outlier cases in which multiple train connections are present, but these are not very relevant. From all edges, 718 edges are considered symmetric, meaning an edge in the opposite direction with the same length exists. So there is a difference in the travel time depending on the direction for more than half the edges.

Graph characteristic		Value
Nodes	total	804
	connected	391
Edges	total	2150
	symmetric	718
	dominated	1063
Degree	outgoing edges	5.5
	unique edge targets	2.7

Table 5.1: General graph characteristics of the graph used to compute detour paths for travellers hindered by maintenance. Symmetric edges are edges which have another edge in the opposite direction with the same length. Dominated edges are edges where another edge with the same source and destination, but a shorter length, is present in the graph. The degree is given with the mean value for all nodes in the graph.

Besides, it can be seen that a node has on average 5.5 outgoing edges, but only 2.7 unique nodes that can be reached. This difference is again explained by the alternative travel edges. An exponential or power law degree distribution was found in multiple public transport and railway networks [61]. The degree distribution of this graph is similar to an exponential distribution. This means that many nodes have a small degree, and a few larger hubs have a substantially larger amount of outgoing edges.

Some basic characteristics of the passenger streams paths are given in table 5.2. These characteristics give some insight into the train travel used for determining passenger hindrance in the maintenance scheduling problem. All origin-destination paths with a number of passengers below a configurable threshold are filtered out during the algorithm. It can be seen that more than three quarters of the passenger streams are filtered out. These are thus only used for computing the final solution quality. Paths with fewer passengers are on average a lot longer, both in the number of edges and in the travel time. The mean length of a path was computed, both with equal weight for every path and weighted by the amount of passengers travelling over this path on the default hour. The weighted mean of the paths is around 28 minutes, which is almost eight minutes shorter than the unweighted mean. This shows that most passengers travel shorter routes.

Default path characteristic		Value
All passenger streams		
Amount (o, d) pairs		20605
Length (num. edges)	mean	5.90
	weighted mean	3.51
Length (min)	mean	61.58
	weighted mean	33.14
Passenger streams above threshold		
Amount (o, d) pairs		4229
Length (num. edges)	mean	4.00
	weighted mean	3.09
Length (min)	mean	36.21
	weighted mean	28.49

Table 5.2: General characteristics of the default paths taken by travellers. Statistics were computed for the full set of passengers streams and the set of passenger streams with amount of passengers above the threshold. Only streams above this threshold are used during the search for a good schedule. The length of a path is given in minutes as well as in number of edges. Besides that, a weighted mean was added, the length of the path was then weighted based on the amount of passengers travelling over that path at a default hour.

5.3 Detour approximation algorithms

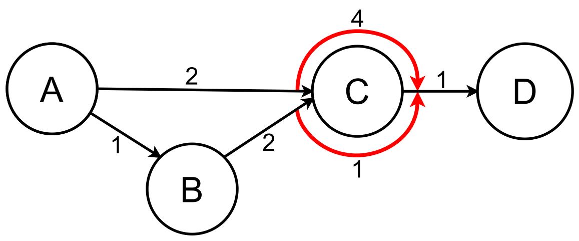
For this research, we have compared different methods to approximate the passenger hindrance computation. Each algorithm assumes as input a set of blocked trajectory parts and a collection of streams. Each stream consists of the number of passengers, source node, target node, and default path. The desired output is the total number of minutes that travellers will have to spend extra in comparison to using the default path.

In this section, six possible ways of determining the detour paths are explained. First, the currently used baseline method is explained. Then, an exact strategy which better handles the passenger's transfer times is presented. An improved version of the baseline will be given after that. Transit node routing is the next new strategy that will be explained. Then, a strategy that assumes passengers always use the alternative transport is given. The last method that is explained is a strategy that tries to approximate the detour paths based on average detours found in a preprocessing step. Besides these six different strategies, we present a few additional algorithmic points.

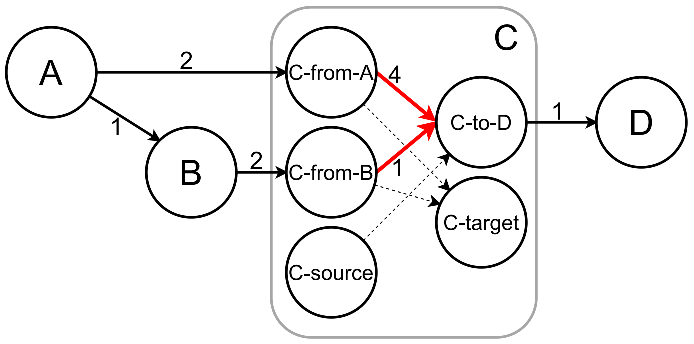
5.3.1 Baseline

In the baseline solution, a single source Dijkstra computation is done to compute the distances from every distinct source in the passenger streams to all nodes reachable. For an explanation of single source Dijkstra, the user is referred to the short explanation in section 4.2. The baseline is the implementation as currently used in production.

This does not always find the absolute shortest path, since this version of Dijkstra only incorporates the transfer time in choosing the second edge of the transfer. This is easiest explained using a simple example, as seen in fig. 5.1a. In this example, the weighted directed graph of 4 stations is given, with an edge for every direct travel connection. The red edges correspond to the transfer times for a transfer from the one edge to another. These are not specified for every node. When finding the shortest path from node A to node D, a Dijkstra algorithm first finds the shortest path to C, which is the direct edge from A to C. Then the final path will be A-C-D, which has a weight of 7. However, the shortest path is A-B-C-D, which has a total weight of 5. So, the basic assumption required by Dijkstra that any shortest path from A through C to D will contain the shortest path of A to C is made invalid by the transfer times.



(a) The graph used by the baseline. The red edges represent the transfer times, these are incorporated when adding the outgoing edge to the path.



(b) The expanded graph as created by the exact algorithm, to allow Dijkstra search to find the exact shortest path. The red edges represent the transfer times. The dashed edges have a weight of 0.

Figure 5.1: An example case to explain the difference between the currently used solution and the created exact solution, where the baseline does not find the correct shortest path from A to D due to the existence of transfer times, and the expanded version of the network.

5.3.2 Exact solution

For the exact solution, the graph used by the baseline is expanded based on the transfer times. These transfer times are given for a collection of (from, through, to) station tuples. Any unspecified transfer is assumed to be length zero. To allow shortest path algorithms to handle these transfer times, the transfer times need to be modelled as edges. The expansion principle will be explained based on a small example graph: fig. 5.1. Any node where at least one transfer time is defined with this node as the through node is expanded. In the example network, only node C has to be expanded. The transfer station node is expanded to create edges for the transfer time. The node is replaced by a set of nodes: one source node, one target node, one ‘from’ node for every incoming edge into the node, and one ‘to’ node is added for every outgoing edge. The original incoming and outgoing edges are then changed from the original node to the corresponding new node. In the example, there are two incoming edges: from A and from B, and one outgoing edge: towards D. So, two ‘from’ nodes and one ‘to’ node are added. Then, one edge from every ‘from’ node towards the target node and one edge from the source node to every ‘to’ node is added. These are the striped edges in fig. 5.1b, with weight 0. These nodes are necessary, since we always want to use the same source node when a passenger starts at this station and, similarly, the same destination node. Finally, an edge is added for every ‘from’-‘to’ combination, with the weight equal to the transfer time of that switch. These are the red edges in fig. 5.1b.

With this construction, the shortest path can be exactly found for every possible graph and route, using the same single source Dijkstra algorithm as in the baseline. However, it does increase the size of the graph and therefore has a negative impact on the runtime of the detour path calculation.

5.3.3 Edge domination

In the graph analysis, it was seen that many edges are dominated. When there exists an edge with the same source and target with a shorter length, this edge will never be a part of the shortest path. By design, there are many of these dominated edges, corresponding to alternative bus travel options. However, the baseline shortest path algorithm still considers these edges while finding the shortest path.

The first new strategy is therefore an improved version of the baseline solution, which reduces the runtime spent on searching for paths over dominated edges. In the baseline solution, these alternative travel edges are also considered when the original edge is not hindered by maintenance. Therefore, an alternative graph is considered, which only considers the non-dominated edges. For every edge that is blocked, the best alternative is added before calculating the shortest paths.

This is done by starting with a preprocessing step before every shortest path calculation, which removes all dominated edges and adds a reference to this dominated edge from the dominating edge. Then, when the blocked trajectory parts are known, every edge which is currently hindered by maintenance is removed from the list of edges. All the corresponding dominated edges are added. These removals and additions are stored, so they can be undone as soon as shortest paths need to be computed with a different set of blocked trajectory parts.

In this first section, this will be treated as a separate solution, to show how much improvement this already gives. However, since this is not actually an approximation compared to the currently used solution in production, this will later just be used as an improved version of the baseline. This strategy will be referred to as ‘Add Dominated When Needed’ (ADWN) from here on.

5.3.4 Transit node routing

The third new strategy for shortest path calculation compared in this section is *transit node routing* (TNR). The railway graph is very structured. Passengers often change trains on the same large hub stations. By incorporating this knowledge with this technique, the shortest paths can be computed more efficiently.

Besides, hierarchical algorithms were successfully applied to similar problems before. The name transit node routing was first used in Bast, Funke, Matijevic, *et al.* [60]. Holzer, Schulz, and Wagner [59] independently developed a very similar technique with more than two levels on the German train network. These papers were combined and adapted to be more suitable for this use case. This strategy exploits the strong hierarchical structure of the railway graph. The important assumption here is that many shortest paths have similar transfer patterns. When looking at the shortest paths in this graph, it can be seen that especially longer paths often go through the same nodes. Therefore, there is also often a lot of overlap between the paths. These overlapping parts only have to be computed once.

This technique was not actually designed to be used as an approximation technique. However, this technique is especially suitable if the same nodes are travelled through on many paths. Every path from a less important node has to go through the same few nodes for most of its paths. The expansion that is done in the exact solution makes the algorithm less successful, since these important hub nodes are split up into many nodes. Therefore, this transit node routing is applied on the baseline graph. This makes it an approximation, which handles the transfer times worse than the baseline algorithm.

To profit from the hierarchical network characteristics, the first step is to determine these ‘important’ nodes. There are many strategies to do this. The most promising ones are explained and compared in section 5.3.4.1. The nodes that are selected as important will be called the high-level nodes, the others will be called low-level nodes. Then, using this set of high-level nodes, we create an overlay graph.

In this research, due to the relatively smaller size of the railway network in comparison with other use cases of this technique, it was chosen to only use a single higher level graph.

This is done by removing the low-level nodes one-by-one and contracting the edges going to or from the node. The procedure for this is described in algorithm 5.1. To merge edges and still incorporate transfer times, edges are extended with a ‘first target’ and a ‘last source’. The first target is the node that should be used to determine the transfer time when transferring to this edge. This is the target of the first edge in the merged edges. The last source similarly represents the source that should be used to calculate the transfer time when transferring towards another edge.

Algorithm 5.1 Contracting the high-level graph.

```
procedure Removing node and contracting graph(Node to remove)
    for Incoming edge towards node to remove do
        for Outgoing edge out of node to remove do
            Merge incoming edge and outgoing edge to create new edge
            Add new edge to graph
        end for
        Delete incoming edges from graph’s edges
    end for
    Delete node from graph’s nodes
end procedure

procedure Merge edges(Edge e1, Edge e2)
    output: Edge merged
    merged.Source ← e1.Source
    merged.Target ← e2.Target
    merged.FirstTarget ← e1.FirstTarget if e1.IsMerged else e1.Target
    merged.LastSource ← e2.LastSource if e2.IsMerged else e2.Source
    T ← transfer time from e1.LastSource through e1.Target to e2.FirstTarget
    merged.Weight ← e1.Weight + T + e2.Weight
    merged.TrajectoryParts ← e1.TrajectoryParts ∪ e2.TrajectoryParts
    merged.IsTrain ← e1.IsTrain ∧ e2.IsTrain
end procedure
```

The next step is to determine which passenger streams will use this overlay graph, and which are local. Any stream for which there exists a path that does not pass through any high-level node is considered local. These are the shorter paths, that are not expected to benefit from the overlay graph. The detour paths for these local passenger streams will therefore still be computed using a secondary regular algorithm on the original graph. The difference is made in the long distance paths, which are split into 3 different parts: an access part, a high-level part and an exit part. Each of these parts can also be empty.

To efficiently calculate the shortest paths, some preprocessing is done. For every low-level node, the potential access and exit nodes are determined. The access points are the high-level nodes that can be reached from the node without passing any other high-level node first. The exit points are determined the same way but in the opposite direction. For every access/exit point, the default access/exit path is computed.

Now, whenever a long-distance passenger stream’s default path is blocked, the detour path is computed. This is done by first recomputing the path from the source to every access point of the source, and from every target’s exit point to the target, if the stored default path for this access/exit is blocked. Besides that, every edge in the high-level graph that is blocked needs to be replaced. This is done by recomputing the path on the original, full graph. Then, for every access/exit pair, the path from the access point to the exit point is recomputed on the overlay graph. The access and exit combination with the shortest combined length of access, overlay, and exit path is returned.

This is illustrated with an example in fig. 5.2. In this case, it can directly be seen that all paths originating from B, C or D will travel through their access nodes A or E, except the paths be going to B, C or D.

These are the local paths. Any paths without origin or destination B, C or D will not have to consider this low-level part of the graph. They will only use the high-level graph. In the high level graph, this low-level part is replaced by an edge from A to E.

If, for example, the edge from C to D is blocked by maintenance, the edge A to E is recomputed on the full graph to. Besides that, the access paths towards E are recomputed for B and C, and the access path towards A is recomputed from D.

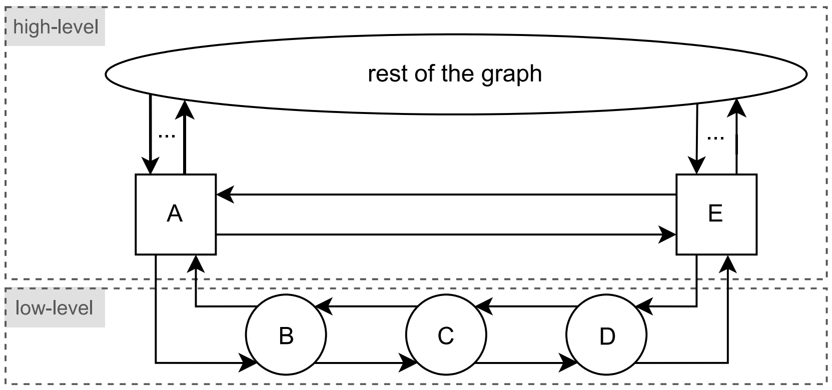


Figure 5.2: An example to illustrate how the transit node routing with the high-level graph and low-level nodes works. The square nodes are high-level nodes. The edge from A to E is a high-level edge, which was created by contracting the original graph and merging the edges from A to B, B to C, C to D, and D to E. A and E are the access/exit nodes for low-level nodes B, C, and D.

This strategy suffers from the same problem of not correctly incorporating transfer times as the original baseline. As previously explained, the baseline only looks at the transfer time in choosing the 'to' edge. However, the transit node routing determines the shortest paths for all 3 parts independent of the transfer times for the transition points: from the access to the high-level and from the high-level to the exit part of the path. Therefore, this will find somewhat worse paths than the baseline algorithm.

5.3.4.1 Determining the high-level nodes

The full comparative results of all strategies will be given in section 5.5. To determine the right nodes in the high-level node, some preliminary experiments were done. Specifically, we have done a full factorial comparison of possible high-level graph construction strategies. This is done with different hyperparameter settings for the criterion, edge filter and high-level size. Each of these hyperparameters and their levels will be discussed. Then, the results of the comparative experiments are given and analysed.

First, the criterion to determine which nodes should be in for the high-level graph. The chosen vertices for the high-level graph strongly impact the result. There are many possible strategies for this. A thorough comparison of eight global selection criteria was done by Holzer, Schulz, and Wagner [59], using real-life railway networks. This research shows the best results are obtained with either a betweenness criterion or a degree criterion for the railway graphs. They also obtain good results with an approximation of the betweenness, but the network size we use does not require approximation.

Manually inspecting the resulting high-level graphs showed that the degree was generally quite suitable, but in some cases, the high degree was not actually related to people often travelling through there. For example, there is one train line which does not always stop at all stations on the route. This causes some stations on that line to have direct connections to a larger number of other stations. However, these stations are on a single line and are rarely a hub for passengers. The betweenness did not have this issue, but had some other drawbacks. Some parts of the Dutch railway network have a very high betweenness, but would not be desirable to have in the high-level graph. The design of the Dutch railway network has one very central point, where many travellers have to go through: Utrecht Central. However, many smaller stations around this one node also have a very high betweenness. This is not desirable, since these stations will not add much use to the high-level graph, since almost all travellers passing through there will also pass through Utrecht Central, which is a better hub to have in the overlay graph.

Based on the good initial results of using the degree criterion on our graph instance and this manual inspection of the results of these two criteria, we have chosen to also include two altered versions. In this altered version, a secondary criterion is used as a tiebreaker in case of equal degree. The first strategy is ‘transfer-degree’, where nodes which have a specified transfer time in the input data are preferred in the case of equal degree. This criterion was added, since the important nodes that are good in the high-level often overlap with nodes where passengers transfer from one train to another. Within the problem definition, these stations are given by having a specified transfer time when this node is the ’through’ station. The second additional strategy is ‘between-degree’, where nodes with the high betweenness are preferred when the degree is equal.

The second parameter is the high-level graph size. If the high-level graph is too small, a large part of the streams is local and can not benefit from the contracted graph. If the high-level graph is too large, the benefit of contracting the high-level graph is lost. We used the same high-level sizes as Holzer, Schulz, and Wagner [59], namely 3%, 5%, 8%, 10%, and 15% of the total graph size.

The final part of the high-level graph construction that is varied is the usage of an edge filter. Whenever two edges are merged, the first target and the last source on the path represented by the contracted edge are also stored (see algorithm 5.1). This is necessary to make sure that the transfer time is computed correctly. After each node removal, edges which are dominated by another edge are filtered. This means that if there are two edges with the same source, target, first target and last source, only the shortest edge is kept. If the additional edge filter setting is used, in the last round of filtering, any edge with the same source and target is considered for domination. This means that only one method of transferring towards this direction is kept. When the transfer time for this direction is high, one of the edges that was removed may have been better. This means that there are fewer edges, resulting in faster computation, although it might result in a suboptimal path on the high-level graph being found.

The complete results of this comparison can be found in table 5.3. Each of these factors has an impact on the results. It can be seen that using only 3% of the vertices in the higher-level graph results in the longest execution times. Some results even reach the time-out, meaning it performs worse than the exact solution. 5% is also too small; results using these high-level graphs are all dominated by other hyperparameter settings. For the remaining sizes, it can be seen that the larger sizes have lower errors, but generally higher runtimes. This same trade-off can be seen in the other hyperparameters. The betweenness criterion for example results in a lower error, but higher runtime. The same applies for the edge filter.

These trade-offs are not always completely even, the results are generally better with the edge filter and with the degree-based criteria. Overall, the differences are not that large between the results of these good sizes (8-15%), with edge filter and some degree-based criterion. From now on we will use 2 distinct hyperparameter settings, which were chosen from this experiment to cover the available range of the runtime-error trade-off. These settings are highlighted in table 5.3. Both will have the edge filter. The first has the size 15%, the other has the size 8%. The first will be slower and more precise, and the second will be faster and less precise. The first will have a degree criterion, where having a transfer time is used as tiebreaker. The second will have a degree criterion, with random tie breaks.

5.3.5 Assume bus strategy

The fourth new strategy for approximating the travel time delays is based on an assumption that the shortest path will often still follow the same path. The passenger uses the provided alternative travel where regular train traffic is hindered. Based on this assumption, the blocked edges on the default path can be directly replaced with an alternative travel edge. This will of course not find available detour paths over a different route and is therefore expected to have worse solution quality. However, the direct replacements can be precomputed for every edge, which can result in very fast detour path computations.

The initial results for this showed that the solution quality was very bad. The main problem was that very long edges were completely replaced by a bus edge. While taking a train to the station closest to the maintenance work, and only using the alternative transport for a smaller section, is faster. An attempt was done to resolve this by attempting a recursive replacement strategy. A longer train edge would first be replaced by multiple shorter train edges, after which only the shorter ones that are blocked will be replaced by an alternative travel edge. To ensure that this still follows the same assumption of using

Hyperparameters			High-level graph			Runtime		Error per stream	
Criterion	Edge filter	High-level nodes (%)	Nodes	Edges	Access nodes	Setup (s)	Test (ms)	Mean (%)	Stddev (%)
betweenness	T	15	121	374	1.76	29	98.56	1.48	4.52
degree	**T**	**15**	**121**	**415**	**1.57**	**27**	**95.47**	**1.34**	**4.48**
transfer-degree	T	15	121	413	1.56	30	93.92	1.43	4.60
between-degree	T	15	121	407	1.55	30	102.36	1.48	4.69
betweenness	F	15	121	616	1.76	34	112.19	1.28	4.30
degree	F	15	121	522	1.57	29	104.52	1.40	4.63
transfer-degree	F	15	121	520	1.56	30	102.07	1.32	4.30
between-degree	F	15	121	518	1.55	28	97.25	1.40	4.63
betweenness	T	10	80	276	1.97	26	90.84	1.42	4.37
degree	T	10	80	299	1.79	22	72.15	1.56	4.82
transfer-degree	T	10	80	301	1.82	22	77.12	1.54	4.81
between-degree	T	10	80	295	1.85	24	77.61	1.49	4.72
betweenness	F	10	80	615	1.97	26	103.80	1.37	4.34
degree	F	10	80	460	1.78	22	77.40	1.49	4.74
transfer-degree	F	10	80	460	1.81	22	76.50	1.49	4.74
between-degree	F	10	80	460	1.85	21	78.12	1.45	4.69
betweenness	T	8	64	230	2.20	26	94.32	1.39	4.34
degree	T	8	64	255	1.96	19	70.95	1.53	4.82
transfer-degree	**T**	**8**	**64**	**261**	**1.94**	**19**	**69.39**	**1.56**	**4.85**
between-degree	T	8	64	252	1.99	19	68.41	1.77	5.09
betweenness	F	8	64	609	2.20	24	101.60	1.33	4.30
degree	F	8	64	487	2.07	21	81.14	1.56	4.84
transfer-degree	F	8	64	453	1.93	20	74.05	1.50	4.75
between-degree	F	8	64	458	1.99	19	72.94	1.60	4.90
betweenness	T	5	40	186	2.81	29	153.46	1.17	4.06
degree	T	5	40	188	2.38	20	78.03	1.79	5.13
transfer-degree	T	5	40	184	2.28	20	79.58	1.67	5.02
between-degree	T	5	40	180	2.34	20	78.23	1.67	5.02
betweenness	F	5	40	627	2.81	29	162.42	1.13	4.02
degree	F	5	40	559	2.61	20	95.14	1.54	4.76
transfer-degree	F	5	40	503	2.43	19	82.47	1.60	4.93
between-degree	F	5	40	487	2.34	20	83.53	1.50	4.82
betweenness	T	3	24	114	3.74	75	>360	1.11	3.64
degree	T	3	24	152	3.88	49	267.44	1.37	4.23
transfer-degree	T	3	24	154	3.38	43	250.18	1.40	4.27
between-degree	T	3	24	148	3.34	42	202.37	1.41	4.28
betweenness	F	3	24	1033	3.74	116	>360	1.08	3.59
degree	F	3	24	688	3.16	28	210.37	1.45	4.53
transfer-degree	F	3	24	878	3.40	50	324.17	1.26	3.80
between-degree	F	3	24	930	3.34	43	207.29	1.31	4.17

Table 5.3: A full factorial evaluation of different hyperparameter settings for generating a high-level graph for transit node routing. The settings chosen for further experiments are boldfaced. These are selected to cover the trade-off between good runtime and lowest error as well as possible.

the same route, this is only done when the shorter train edges use a subset of the route of the edge to replace. This is necessary since we want to have a fixed precomputed replacement per edge. If it is not enforced that these are sub-edges, this can result in cycles in the recursive replacement. These additions did not improve the results sufficiently, as the shorter edges were not always correctly found. Besides, the case that most of the longer edge is blocked is also regularly occurring. In this case, splitting it into smaller edges before replacing the edges with alternative transport results in a longer path than just replacing the full edge.

It may be possible to resolve this by precomputing both replacement possibilities, replacing with shorter train sub-edges and with a full bus edge. Then, based on the number of sub-edges available, the right replacement is chosen. This would increase the complexity of the strategy, which was designed to be very simple, even more. Besides, the resulting path would still often be far off. Another downside of this method is that it will not help to prevent maintenance from being scheduled on a common detour route for passengers.

These results led to the conclusion that the same path assumption has too many drawbacks, and it is difficult to successfully use as a strategy. Since another fast implementation yielded better initial results, this strategy was not experimented with further.

5.3.6 Average delay strategy

The fifth approximation strategy we developed is an average delay strategy (AD). An average amount of extra travellers minutes is precomputed for every edge in the graph. This strategy was explored to find a solution in the lower end of the runtime range. However, based on the results of the ‘assume bus’ method, a strategy was required that incorporates whether there are alternative train routes available. Therefore, the delay is based on which specific edge is blocked and not just the length of blockage.

For the total cost computation, it is also necessary to know how many passengers use the alternative travel and on which subcorridors they use these buses. Therefore, to make a constant delay per edge strategy feasible, it needs to precompute for every edge, the expected travel time delay as well as the expected alternative travel behaviour. a percentage of the passengers which will be using the bus and on which subcorridor we want to add these bus users.

To precompute this average delay, we compute the hindrance of scheduling all the project requests one-by-one. The extra traveller minutes are divided over the blocked edges of the default path based on the length of the blocked edge. The same is done for the bussed subcorridors. Take for example a passenger stream with 10 passengers with a default route blocked on two edges, A and B. Edge A has length 20 and edge B length 10. If the detour path takes 6 minutes longer, and uses the bus on one subcorridor. Then a blockage of edge A is stored to have an expected delay of 4 minutes and two thirds of the passengers will use the bus on the subcorridor. Blocking edge B receives an expected delay of 2 minutes and one third of the passengers using a bus from this stream. When this preprocessing is done based on many streams and averaged based on the amount of passengers on the pre-processed streams, we are able to estimate how important every edge is and whether there are easy possibilities for different routes.

This loses a lot of accuracy. Especially the mapping between which edge is blocked and which subcorridors will have bussed passengers is far from optimal. However, no shortest path computations are necessary during the search, making it much faster than the previous implementations.

5.3.7 Additional algorithmic points

Two additional points, concerning the approximation techniques presented before, are explained in this section. First, an improvement based on symmetry is presented. Then, an explanation of the bias present in some methods is given.

5.3.7.1 Exploiting symmetry

An important observation that was made during the problem analysis is that the difference between a path from station A to station B and from station B to station A is generally quite small. Therefore, another approximation was done by combining these two passenger streams. The new stream consists of passengers equal to the sum of the number of passengers of the two. The length of the default path of the new stream is the mean length of the default paths of the two streams.

To distinguish between passenger streams with the same source and target following a different default path and streams following the same path, only streams that require the same trajectory parts are combined. This will prevent cost differences caused by the original passenger stream being blocked, whereas the new passenger stream is not, and vice versa.

To reduce the amount of shortest paths that will have to be computed, it is most important to reduce the number of distinct sources. This is the most important factor determining the possible speed up, since the objective evaluation uses a single source Dijkstra strategy. Therefore, we use a simple greedy precomputation to determine in which direction the detour path should be computed for each stream. This means going over all streams one by one. If either the source or the target is already in our set of chosen sources, this one in used as source. Otherwise, the one with the lowest unique identifier is used as source.

This results in a symmetrical version, which has 2101 streams rather than 4229 streams with the default threshold. It has 241 distinct source nodes, compared to the original 357 distinct source nodes. This strategy can be applied in combination with each of the aforementioned strategies, since it works on the passenger streams rather than the actual shortest path computation.

5.3.7.2 Scaled versions of the detour path approximation techniques

Some of the proposed techniques are always overestimating the delay of the passengers. Both the baseline and the transit node routing technique are based upon using actually feasible routes. Time is saved by accepting that sometimes non-optimal routes are chosen. So, each result is either equal to or longer than the exact path. This means that the passenger costs are always overestimated.

The alternative travel costs can either be overestimated or underestimated. The non-optimal path could either be a route without alternative travel, while there is a faster detour route available that uses the provided buses. Or, the suboptimal algorithm may choose a route with a bus, while there is a faster detour route which only uses train edges.

This general overestimation of delay can negatively impact decision-making. When the algorithm needs to make a decision between higher passenger hinder costs and higher costs in a different part of the cost function, it may sooner choose to incur costs in different parts when using an overestimating approximation technique. However, the exact path computation is actually an underestimation of the hinder costs, since streams below a configurable threshold are not considered.

A different downside of the overestimation is that when comparing between different detour path approximation techniques, the reported error is difficult to interpret. If all paths are equally overestimated, this may result in better choices during the scheduling process, as compared to paths that have more uneven errors. The choices between different starting times, with different passenger hinder, is more likely to be optimal with even overestimation.

For these two reasons, some experiments were also done with a version of the approximation techniques that linearly scaled the approximation technique. The results were similar for the different approximation techniques. And since the impact on the complete schedule was very small, it was chosen not to continue with this scaling, since the hard-coded changing of the path lengths is not very desirable from a code quality perspective.

5.4 Methodology for comparing detour path approximations

To compare the different strategies, it is important to quantify the trade-off in runtime and solution quality. The impact on the solution quality can be measured on different levels. Three different levels were used in this work. Each of these levels contribute to evaluating a method. For all of them, five replications are done.

The first level we consider is the error in travel delay per stream. This is the lowest level we use, which gives the most direct and easy to interpret results, as it gives a feeling of the amount of minutes travel-time the solution is off. Besides, we can see the error independent of any aggregation. However, since the symmetric version does not have the same passenger streams, this can not be used for these versions. Since only computing a single path has a very short runtime, this will not be measured on this level.

Second, the complete cost of expected extra travel time and alternative transport for one hour are considered. This level is important, since it allows evaluation of the symmetric versions. Besides, it gives a measure of all impact from the approximation, by including the alternative transport. The alternative travel costs are also dependent on the found detour paths. Methods which have a bias will perform worse on this. If, for example, an approximation method always overestimates the length of a path, this will have a larger aggregated error. However, overestimating may not necessarily perform worse in making the best scheduling decisions. The runtime is measured for the computation of these costs, for updating one hour, without using any cached paths.

Last, the most promising strategies on the first tests are used in a complete greedy scheduling run to quantify the impact on a complete schedule. The full runtime is measured for creating and evaluating a schedule with the greedy algorithm.

5.4.1 Separate test set

To evaluate the performance of different methods separately from the rest of the algorithm, an isolated test set of realistic detour paths to compute is required. Therefore, a large test set was created from that could be executed separately from the scheduling to see the impact on paths, resulting costs, and runtime the new method has. This test set is used for comparing the detour path approximation strategies on the most precise levels.

This isolated test set was created by randomly extracting the information of 0.5% of availability cost calculations in a greedy algorithm. For each availability cost calculation, the set of detour paths and the hour were stored. Then. each of the detour path approximations can be used to find the shortest path, and compute the error from the exact shortest paths.

5.5 Results

First, the results from the isolated test set are given, both on a path level and an hourly level. Then, the results of applying the approximation to create a full schedule are provided.

5.5.1 Detour path approximation on isolated test set

The results of experiments to compare the detour path approximation techniques on the isolated test set are presented in this section. The most specific path level results are given in table 5.4. The results of the complete hour test set are given in table 5.5. The mean runtime and approximation errors on both levels are also shown more visually in fig. 5.3.

Method	Path length error	
	Absolute (min)	Relative (%)
Exact	**0.0**	**0.0**
ADWN/Baseline	0.5	0.8
TNR1	0.8	1.4
TNR2	0.9	1.5
AD	9.6	20.4

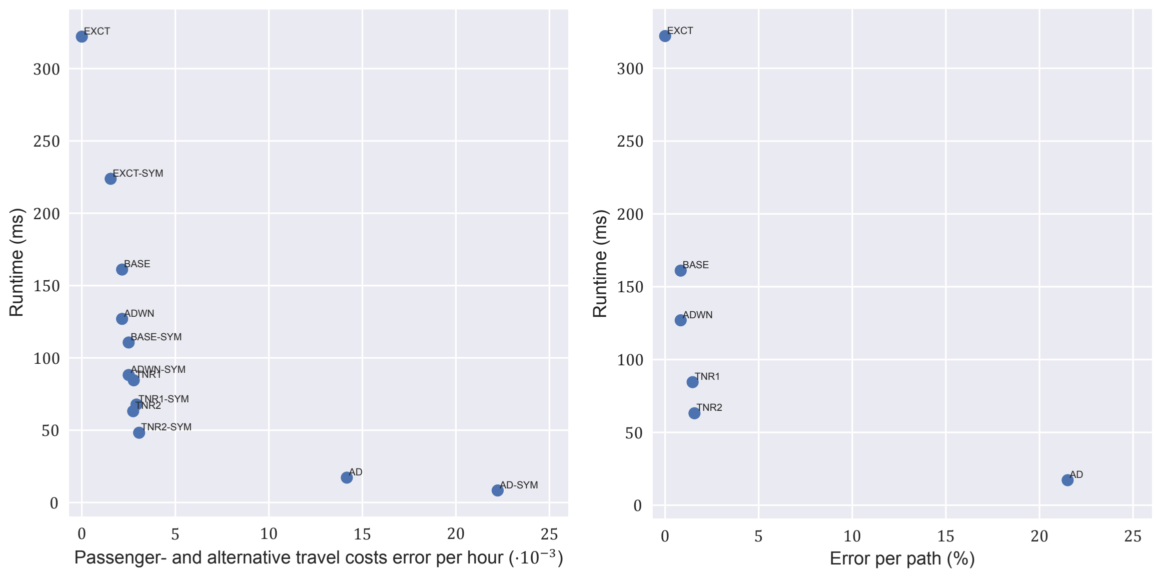
Table 5.4: Five repetitions of computing all detour paths from the isolated test set with the detour path approximation algorithms. The best result is boldfaced. ADWN is the improved version of the baseline. TNR is transit node routing, with two different hyperparameter settings. AD is the average delay strategy. The details of these techniques are explained in section 5.3.

When analysing the results of the path length error, it can be seen that the average delay has the largest errors by far. These errors are spread out, some paths are more than twice as long as the exact detour path. The length of the paths found using the average delay strategy are on average over 20% different from the exact shortest path, which is quite high. This shows that this average delay per edge is quite a large simplification, which was expected.

All other approximations have a mean error of less than a minute for the approximated paths. The difference between the two hyperparameter settings of the transit node routing is very small. These are very small differences, and it is not expected that this will have a large negative impact in the

Method	Runtime (ms)		Cost error	
	Mean	Stddev	Absolute ($\cdot 10^{-3}$)	Relative (%)
Exact	322.1	26.2	**0.00**	**0.00**
ExactSymm	223.8	19.6	1.46	1.67
Baseline	161.1	14.7	2.04	2.26
BaselineSymm	110.6	10.9	2.37	3.04
ADWN	127.0	11.4	2.04	2.26
ADWNSymm	88.1	8.6	2.37	3.04
TNR1	84.5	9.4	2.63	3.04
TNR1Symm	67.7	7.8	2.77	3.52
TNR2	63.1	9.3	2.60	3.00
TNR2Symm	48.2	6.5	2.90	3.57
AD	17.2	2.7	13.43	21.02
ADSymm	**8.3**	1.4	21.08	26.76

Table 5.5: Five repetitions of the isolated test set for hourly passenger availability cost computations with different detour path approximation techniques. The best result for each column is boldfaced. ADWN is the improved version of the baseline. TNR is transit node routing, with two different hyperparameter settings. AD is the average delay strategy. The details of these techniques are explained in section 5.3. The Symm techniques uses the symmetrical version of the passenger streams.



(a) The error of different detour path approximations per hour on an isolated test set. The mean difference in the sum of the passenger and alternative travel costs is used.

(b) The error of different detour path approximations per stream on an isolated test set. The mean relative difference of length per origin, destination pair is used.

Figure 5.3: The error of different detour path approximation techniques on an isolated test set. The runtime is the mean time spent on running the isolated test set, which is one hour of calculating the total extra travellers minutes and the resulting passenger hinder and alternative travel costs. The x-axis shows two different measures of the quality of the approximation. ADWN is the improved version of the baseline. TNR is transit node routing, with two different hyperparameter settings. AD is the average delay strategy. The details of these techniques are explained in section 5.3. The Symm techniques uses the symmetrical version of the passenger streams.

scheduling process. Most paths were perfectly found, many of the mistakes were only a few minutes. A few outliers had larger errors. This happened especially at paths where the defined transfer times were high.

The results of the hourly test set shows that the symmetrical solution has a clear positive impact on the runtime. The symmetrical strategy also results in an increase in cost error. These errors are small enough that the symmetrical versions often result in a new non-dominated solution.

The improved version of the baseline performs as expected, the same paths are found as the baseline, but the runtime is significantly lower. The average delay still has the largest errors, but is also more than twice as fast as the fastest alternative in recomputing the passenger hindrance for one hour. The difference in the error between the two versions of the transit node routing is even smaller on the hourly results. The transit node routing 2 with a smaller high-level graph has a faster computation of the availability costs.

From the plot of the errors as compared to the runtime (see fig. 5.3), it can be seen that the solutions forms a convex Pareto front. A Pareto front is a commonly used method to visualise and analyse different solutions in a multi-objective problem. In fig. 5.3, also the Pareto-dominated solutions are kept, but the shape can still be analysed. The runtime and the cost error can be seen as two objectives. These two objectives are opposing. Decreasing the runtime comes at a certain cost to the error. From the convex shape, it can be seen that the largest part of the runtime can be decreased by only incurring a limited error. The last bit of improvement is most expensive. There are multiple, non-dominated approximation techniques found, which can be beneficial for trading off solution quality and runtime.

5.5.2 Impact on complete schedule

The results of the isolated test set were used to determine the most promising techniques. These techniques were then applied to the full scheduling problem. Only one hyperparameter setting for the transit node routing was continued with. As can be seen in fig. 5.3a, the TNR1 approximation was very close in runtime to the symmetrical version of the improved baseline, but with higher mean error. The symmetrical version of TNR1 was dominated by TNR2, meaning it was both slower and with worse solution quality. Therefore, only TNR2 is used, which will from here on be called TNR instead. The old baseline was slower than the improved baseline (ADWN) approximation, with exactly equal detour paths being found. The old baseline path computation will therefore also not be used in the comparison of the impact on the complete schedule.

For the remaining four techniques, both the symmetrical and the regular version were applied to the full problem. The runtimes and quality of the resulting schedule solving are provided in table 5.6. There are no clear differences in number of constraint violations between any of the approximation techniques with this number of replications. This is as expected; even if slightly worse decisions relating to the passenger hinder are made, it is not expected that this has a direct impact on the number of broken constraints. When a constraint violation can directly be prevented, this always has precedence over reducing the cost in the greedy scheduling algorithm.

The result that stands out the most is the average delay strategy. This strategy results in a worse solution quality. The errors that were found in the isolated test set result in slightly worse scheduling decisions being made. This difference is only around 13.3, which is approximately 1%, but that is still a cost difference that is considered important for the schedule quality. Another interesting result for the average delay strategy is the fact that this strategy is not the fastest, which it was on the isolated test set. The paths found by the average delay strategy generally require more subcorridors where alternative travel needs to be provided. One passenger stream can result in many small amounts of passengers using different buses. This makes calculating the alternative costs based on this path a bit slower. This alternative travel calculation has to be done very often, also when re-using the path from the cache. This is why this has a stronger impact on the runtime in the complete schedule, compared to the isolated test set. In the end, this made it not competitive with other approximation methods with a higher quality and faster runtime. This shows the importance of speeding up this alternative travel costs handling, in order to find a faster approximation.

Detour approximation technique	Cost		Hard constraint violations		Runtime	
	Mean	Stddev	Mean	Stddev	Mean (hh:mm)	Stddev (mm:ss)
Exact	1107.46	1.48	4.8	2.5	03:19	05:40
ExactSymm	1109.08	0.47	**3.6**	0.6	02:23	02:53
ADWN	1107.87	3.67	5.2	1.8	02:02	02:00
ADWNSymm	1109.28	0.47	5.6	1.3	01:32	01:23
TNR	**1106.53**	5.36	5.0	0.7	01:44	02:45
TNRSymm	1108.28	5.00	5.0	1.2	**01:22**	01:14
AD	1119.93	0.28	4.4	2.8	02:10	04:38
ADSymm	1121.67	1.89	3.4	0.9	01:32	02:20

Table 5.6: Five repetitions of a greedy scheduling run for the year 2024 with different detour path approximation techniques. The best mean results are highlighted for each column. ADWN is the improved version of the baseline. TNR is transit node routing, and AD is the average delay strategy. The details of these techniques are explained in section 5.3. The Symm techniques uses the symmetrical version of the passenger streams.

For the other approximation techniques, the runtime differences are more similar to the results from the isolated test set. Exact is clearly the slowest algorithm. Transit node routing is the fastest. The improved version of the baseline (ADWN) is between these two. The differences are of course smaller than on the isolated test set, since only a part of the execution is influenced by the shortest path calculation. Whether the symmetrical strategy is used makes relatively more difference in the full run. When the symmetrical version is used, this also speeds up calculation when all paths are already present in the cache, since fewer paths have to be accessed and used in cost computation. The symmetrical version of the improved baseline is therefore faster than the non-symmetrical version of TNR when using a full schedule run, whereas it was the other way around on the isolated test cases.

The mean costs for all approximation techniques except the average delay based strategies are all very close. No strategy clearly outperforms the others. This shows that the symmetrical version of the transit node routing does not result in a large solution quality loss, while being substantially faster, compared to the exact detour path calculation.

5.6 Conclusion

Multiple approximation methods are possible, with small differences between the detour paths found by the exact method and the approximation technique. Most paths were the same, when a different path was found, it was often only a few minutes longer. Only a few paths were further off, when the defined transfer times were high. The developed techniques show a trade-off between runtime and solution quality on an isolated test set.

When applying those techniques to solve the full scheduling problem, the average delay approximations were dominated by other techniques. The faster computation did not hold up for a full scheduling run, due to too complex handling of the alternative travel costs. However, if a better handling of the bus costs could be found, it shows that a potential approximation which has mean errors of up to 30% for this part of the objective, the loss of solution quality is still less than 2%. All other techniques resulted in a similar final solution quality, where the symmetrical version of a transit node routing was fastest. This shows the possibility to solve the year 2024 with negligible loss of solution quality in less than half the time that would be required to solve it with an exact solution.

Using these results, the research question: “How can approximation of passenger detour paths be used for faster solving?” can be answered. Approximation of this part of the objective is an effective way to reduce runtime, with negligible impact on the solution quality. By using methods to find short paths that are not necessarily the shortest path, there was only limited error in the path length and in the resulting passenger availability costs. This error is small enough that it has no detrimental effects to the decisions made by the greedy scheduling algorithm, and find equal quality solutions.

A critical note that should be added to these results is that there is a downside to the usage of the exact passenger streams. These paths are used for comparison to determine the errors in this chapter, as well as for the final costs. The assumption is made that passengers always take the shortest path on the modelled graph. There are two problems with this. First, passengers do not always use the shortest path. They can have other preferences that supersede the travel time in some cases, such as fewer transfers [62]. Second, the travel time graph is not a perfect representation of reality, and the resulting shortest path can sometimes show weird choices. The main reason here is the addition of transfer times. Having some penalty when passengers switch trains makes it more realistic, however these transfer times are not given for all stations. When nothing is specified, this is interpreted as having no transfer time. This causes shortest paths to sometimes take stranger routes, which are faster only due to having no specified transfer times.

5.6.1 Future work

The work done in this chapter has given rise to a few potential future research directions that could improve the solving of the railway maintenance scheduling further, using the approximation of the passenger detour paths.

The most important future research direction is to look further into finding a smaller, representative subset of passenger streams. Merging paths that took the same path in different directions resulted in significant speed-ups. An important advantage of this approximation is that it does not only reduce the number of detour paths that needs to be computed, but it also reduces the effort of calculating the hinder when the detour paths were already calculated earlier. The first suggestion for further decreasing the number of streams, and specifically the number of sources, would be to group streams based on having many edges in common. Then, using the most average stream as representative for that group, preferably with as many streams with the same origin, could reduce runtime further. However, other ways of merging passenger streams or even merging stations and their corresponding nodes in the graph could be interesting future research possibilities.

In the other used detour path approximation strategies, some improvement possibilities were also encountered. The transit node routing could be further improved by more research into caching the (partial) paths. The average delay approximations may be improved by simplifying the alternative travel cost approximation. From the results on this technique, it would be interesting to approximate the cost based on average delays for a complete hour, rather than averaging bus costs per path. This could give more reliable results, and reduce the runtime of computing the costs a lot further. This hourly approximations could also be used outside the average delay strategy; approximation per hour instead of per path has potential to obtain more efficiency improvement than a path approximation.

Another future research direction is determining related trajectory parts. Whenever a different set of trajectory parts is blocked, new detour paths are computed for all passengers. However, passengers travelling a detour path in the northernmost part of the country, will most likely not be hindered by maintenance in the south. So, whenever there is a second request scheduled simultaneously, but geographically far apart, all detour paths are recomputed, despite it probably not affecting most of the existing hindered passengers. By precomputing for every passenger stream which trajectory parts may be relevant in a possible detour path, this situation could be prevented. In that case, only the overlap between the related trajectory parts and the blocked trajectory parts is considered for reusing paths.

The results from this chapter also show that if a different part of the objective has a trade-off comparable to this one, where a large part of the runtime can be reduced by incurring a small error in the objective, this could be beneficial for the solving. This principle could be further applied to both this problem, and the specific strategies proposed here could be applied to other expensive optimisation problems involving detour path calculations.

The final future work direction would be to use a more realistic model of the passenger behaviour during maintenance. As concluded previously, more realistic modelling of the passenger hinder is necessary to fix the problems in some strange paths being found due to wrong transfer times, and to be able to more correctly evaluate approximation strategies. Improving the transfer time modelling would therefore be the first recommendation. Another aspect that could be considered here is whether passengers will still travel when maintenance occurs. Part of passengers will delay their journey, depending on the moment.

6 Search Strategy Comparison

Having different search strategies available to create a maintenance schedule is an important way to make a larger part of the trade-off between runtime and solution quality available for the ProRail scheduling team. After the general optimisations explained in section 4.4 and the detour path approximations explained in chapter 5, computing the objective is faster. The greedy algorithm that took around twenty-four hours at the beginning of this work can now be run in around an hour and a half. Therefore, the strategies proposed by Oudshoorn [5] are feasible again. The existing search strategies are explained briefly in section 6.2. Then, a new look-ahead addition to the greedy constructive algorithm is proposed in section 6.3. All strategies are compared based on solution quality and runtime in section 6.4. Finally, the research question “How do different search strategies solve the scheduling problem in terms of solving time and solution quality?” is answered based on these results.

6.1 Background

In this section, the necessary background is given for the discussed search strategies in this chapter. The background on greedy algorithms was provided in chapter 4. The background concerning the evolution strategy and the proposed look-ahead technique will be given here.

6.1.1 Evolution strategy

Evolution strategies (ES) are a large family of nature-inspired algorithms, and one of the common types of evolutionary algorithms. Originally, there were 2 main rules for ES: 1) change all variables at a time, mostly slightly and randomly and 2) if the fitness does not decrease keep the new individual, otherwise keep the original [63]. Besides this, one of the main features of an ES is the self-adaptive mutation strength [63]; the size of the mutations is changed based on the current success rate.

This has been extended with larger populations and different ways to select the new population [63], such as tournament selection. Recombination was also added in some versions of ES. Mutations used in evolution strategies, which generate offspring from a parent, are problem-dependent [63]. The mutations have become increasingly de-randomised in newer variants of the evolution strategy [64]. Evolution strategies have been successfully applied to numerous optimisation problems, and are still continuously being developed further [64].

6.1.2 Look-ahead techniques

The main idea of look-ahead based techniques is to look forward towards the next steps of an algorithm, before making the current decision.

Applying this look-ahead in combination with a greedy algorithm is explained in the work by Chen [65]. This combination has been successfully used for solving multiple problems, such as job shop scheduling and circle packing [65], [66]. More recently, a similar principle was applied to the VNF-FG embedding problem [67], where decisions within the horizon h are used. This improves the solution quality and allows trading off runtime and solution quality based on the value for h.

The same general principle can also be seen in other solving methods besides greedy algorithms. In some value ordering heuristics for constraint satisfaction problems, the number of conflicts occurring with the current value is counted [68]. The least conflicting value can then be tried first. Look-ahead strategies have also been researched extensively in game theory, especially to the occurrence of pathological situations, where looking ahead is counterproductive [69].

An example of look-aheads being applied in research to public transportation is given by Pätzold, Schiewe, Schiewe, *et al.* [70]. Here, they add three enhancements to include an approximation of the cost in the next stage using a simple upper bound. These enhancements allow them to find significantly lower costs for line planning, timetabling and vehicle scheduling for integrated planning in public transportation.

6.2 Existing algorithms

In this section, the algorithms proposed by Oudshoorn [5] are explained: a greedy heuristic algorithm, an evolutionary algorithm, and a hybrid greedy-evolutionary algorithm. The hybrid version provided the best results on the schedule of 2020, with an older version of the problem definition.

6.2.1 Greedy constructive algorithm

The first solving method is a greedy constructive algorithm. This is a simple, but effective algorithm, which is considered the baseline for solving, as it is currently the only algorithm actively being used by the ProRail scheduling team. This method was explained in more depth in section 4.2.

This algorithm assigns a starting time to the requests one-by-one. This is done by trying out each possible starting time with a specified granularity. This granularity is by default daily. Each day is tried and the lowest cost starting moment is assigned. This way, a schedule is created greedily. This algorithm is relatively fast, since a request is not moved after it has been assigned a start time. However, the exploration of this method is limited.

To plan clusters correctly, whenever the current request is a part of a cluster, the full cluster will be added to the schedule. This is done by first finding the longest request(s), and trying all its start times. For every start time, the other requests from the cluster are scheduled greedily within the period of the longest request. The longest request is thus scheduled greedily based on the cost of adding all cluster requests in this time window.

6.2.2 Evolutionary algorithm

Another existing strategy is an evolutionary algorithm, which was developed for solving the maintenance possession scheduling for the Dutch railway network [5]. Oudshoorn [5] presents it as an evolution strategy (ES), although it does not contain the concept of self-adaptation of step size. This self-adaptation is considered one of the main features of evolution strategies [71]. The algorithm may therefore be more adequately described as the more broad term evolutionary algorithm or a population-based local search. In this thesis, we will call it an evolutionary algorithm (EA).

In this algorithm, each chromosome represents a complete schedule as the list with a starting time for each of the requests. The starting population is initialised using a simple heuristic. Then, for a certain number of generations, offspring O is generated from the current population. A rank-based selection is used, which selects the best n individuals from $P^t + O$ for the next population P^{t+1}. To create this set of offspring, m mutations are done. For each mutation, a parent is randomly selected from P^t, with uniform probabilities.

Using the evolutionary algorithm, the found solutions for the year 2019 and 2020 were slightly worse than the greedy constructive algorithm [5]. However, it could be seen that this algorithm performed well on different aspects than the greedy solution. The evolutionary search was better in reducing the availability costs.

Due to the updated problem definition and the slower objective function, the realistically possible number of generation is lower than it was at the time of this original work. Besides that, the mutations were designed back then. These have not been updated to result in a competitive solution quality. The EA is still included in the comparison from this chapter, because seeing how the EA performs on this new problem formulation and input can help to evaluate the hybrid algorithm.

6.2.2.1 Mutations

Each of the mutations is explained briefly in this section; for more detail, the reader is referred to the original work by Oudshoorn [5]. The mutations are separated in three groups. The first group contains *shift* mutations, which randomly shifts blocks. The second group contains *bucket* mutations, which try to group requests effectively. The last group contains *fix* mutations, these try to resolve conflicts and dependency constraint violations. Each of these groups are selected with equal probability. Then, a mutation is selected from the group following a predefined distribution.

Shift mutations. The *shift* mutations are the simplest types of mutation. These mutations do simple shifts to a different start time. The first two types in this group are the *day shift* and *hour shift* mutation, selected with probability of 0.2 each. These take a random request not in any bucket and move it to a random day in the year, or a random time, preserving the selected day. Similarly, the *bucket day shift* mutations are selected with probability 0.25, and the *bucket hour shift* mutations, with probability 0.15. These do the same shifts for a random bucket. Lastly, there is the *multiple day shift* mutation type, which moves ten requests at once to a different day.

Bucket mutations. A bucket is a set of two or more requests that are grouped. A bucket will always be scheduled overlapping, with the goal of decreasing availability costs. In other mutations, all requests in a bucket will be moved simultaneously to keep this overlap intact. A bucket must consist of requests which are closely related, but not conflicting. This means that they can be scheduled simultaneously and that they block some of the same subcorridors. The reason for this is that cost is mostly reduced when these subcorridors have to be blocked only once for all maintenance in a bucket. There are three types of bucket mutations.

A *create bucket* mutation selects two requests that are currently not in a bucket and creates a new bucket from those two requests. The bucket is planned at the starting time of one of the requests with equal probability. Possible pairs to create a bucket from are precomputed. Specifically, two requests can be in the same bucket if the number of overlapping subcorridors is higher than the number of non-overlapping subcorridors of the request of the least impacting request. For this comparison, impact is determined based on length multiplied with number of affected subcorridors. The *create bucket* mutation is chosen with probability 0.4 from the bucket mutation group.

The second mutation within this group is the *expand bucket* mutation, which is also chosen with a probability of 0.4. This mutation selects a random request which is not a part of any bucket and tries to add it to a bucket. This is retried a configurable amount of times if no suitable bucket is available. A request can only be added to a bucket if the number of non-overlapping subcorridors times the length is smaller than the sum of the overlap length between the overlapping subcorridors.

The final mutation in this group is the *shrink bucket* mutation, which is selected with the remaining probability of 0.2. This mutation picks a random bucket and removes one of the requests from the bucket. This request is assigned a random new start time.

Fix mutations. The final group of mutations is the *fix* mutations. These are designed to resolve hard constraint violations.

The first type is the *fix bucket* mutation, chosen from this group with probability of 0.5. This searches for a bucket breaking a hard constraint and finds all available periods where it would not cause any constraint violations. It randomly chooses one of these available periods. Periods that do not require a different time are preferred. If no conflict free periods are available, the bucket is randomly shifted instead.

The second mutation in this group is a *fix conflict* mutation, which is chosen with a probability of 0.25. It selects a random project request not in a bucket which is currently causing a conflict, and moves it according to the same rules as the bucket fix mutation.

The final mutation is the *fix dependency* mutation, which reschedules a request not in a bucket that is causing a dependency constraint violation in the same way.

6.2.2.2 Constraint cooling

The evolutionary algorithm was improved with the addition of constraint cooling. At the start of the search, a certain number of hard constraint violations are allowed. This strategy allows the algorithm a bit more exploration space. This exploration space is important to reduce the cost. These hard constraint violations can then be fixed at later iterations.

This principle will not be explained in depth, since the results of initial experiments showed that this constraint cooling was not influencing the results of the evolutionary algorithm. With the current mutations and amount of generations possible, the individuals always had more broken constraints than the allowed amount of constraint violations.

6.2.3 Hybrid greedy-evolutionary algorithm

When analysing the schedules for the years 2019 and 2020 created by the evolutionary algorithm and the greedy algorithm, it could be seen that the EA was better at decreasing the availability costs [5]. Besides that, the evolutionary algorithm was also better at doing more exploration in complete schedules. Contrarily, the greedy algorithm was better at decreasing the soft constraint penalties, and finding decent solutions fast. This was the motivation for designing a hybrid algorithm [5]. This search strategy obtained the best solution quality for the 2019 and 2020 schedules, with a runtime of twenty-four hours.

This hybrid algorithm uses the greedy heuristic with randomness to create starting individuals for an evolutionary algorithm. Besides that, it splits the evolutionary search in multiple phases. In each phase, it first assigns only part of the requests a starting time with the greedy heuristic. Then, it applies the evolutionary algorithm to improve this partial schedule. A flowchart of this process is shown in fig. 6.1.

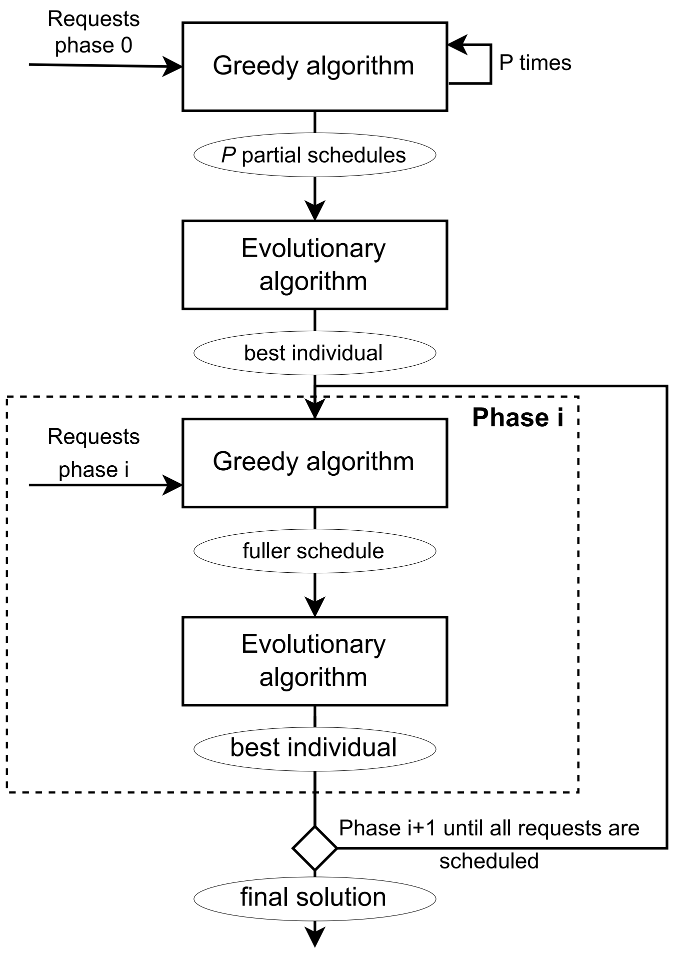


Figure 6.1: Visualization of the hybrid greedy-evolutionary algorithm with a population size P.

The best results were obtained by Oudshoorn [5] with two phases, so that is the setting that will be considered here. The population size, offspring size and number of generations were all decreased in comparison with the original work, to adapt for the new, slower objective evaluation. All hyperparameters are given in table 6.1. It was decided not to use constraint cooling in this strategy. Some first results showed that constraint cooling was not working with the new problem definition and the 2024 problem input. The number of constraint violations was higher at the beginning, as expected, due to the higher allowed constraint violations. However, within the generations that were allowed, some of these new violations were not resolved during the cooling process. The final result of the evolutionary search using constraint cooling therefore had more constraint violations than the starting population.

Parameter	Value
Population size	10
Offspring size	30
Number of phases	2
Requests per phase	(100, N-100)
Generations per phase	(5000, 2500)
Constraints allowed at start	0

Table 6.1: The default hyperparameter settings used with the hybrid greedy-evolutionary algorithm.

6.3 Greedy constructive algorithm with look-aheads

A look-ahead addition was implemented as a possible improvement to the greedy constructive algorithm. With this look-ahead, the impact of the chosen starting time on some still-to-plan project requests will be considered when greedily scheduling requests. To do this, the algorithm plans a few extra requests for every considered starting time. So, in a manner similar to the improved handling of scheduling a cluster, the request is no longer scheduled only based on its own incurred cost. It also includes the information of some other costs that have a high interaction with this request. If, for example, a request with a large time window has two possible starting times with only a small difference in cost. When another, conflicting, request could for example only be planned on the first starting time, it would be better to choose the second starting time for the first request. If this differentiation can be made by including the effect on the second request in the first decision, this may improve the final solution quality.

To demonstrate the process of the greedy constructive algorithm with a look-ahead, two examples are used. In example 1, an example of how a look-ahead can prevent a conflict, as compared to a regular greedy run, is shown. Example 2 shows how unnecessary conflicts can not always be prevented by the addition of a look-ahead. This happens when the number of requests that need to be planned on their cost suboptimal location to prevent the conflict is higher.

Example 1. Two conflicting requests, global available time window 00:00-02:00, greedy schedule order:

- Request A, time window: 00:00-02:00, length: 1 hour
- Request B, time window: 00:00-01:00, length: 1 hour

Desired solution: 0 constraint violations. Request A is scheduled at 01:00-02:00 and request B at 00:00-01:00.

Without look-ahead: 1 constraint violation. Request A will be scheduled at 00:00-01:00. In this example, this period results in a lower passenger hindrance cost. Then, request B will be scheduled, this request has to be scheduled at 00:00-01:00, so there will be one constraint violation.

With look-ahead: 0 constraint violations. Request A will be scheduled at 01:00-02:00. Both start times are tried. Although 00:00-01:00 has lower cost, look-ahead shows it will cause a constraint violation later on. After this, request B will be scheduled at 00:00-01:00.

Example 2. Three conflicting requests, global available time window 00:00-03:00, greedy schedule order:

- Request A, time window: 00:00-03:00, length: 1 hour
- Request B, time window: 00:00-02:00, length: 1 hour
- Request C, time window: 00:00-01:00, length: 1 hour

Desired solution: 0 constraint violations. Request A should be scheduled at 02:00-03:00, request B at 01:00-02:00 and request C at 00:00-01:00.000

Without look-ahead: 1 constraint violation. Request A will be scheduled at 00:00-01:00, since this results in the lowest passenger hindrance cost. Next, request B will be scheduled at 01:00-02:00, since this is the only start time for request B which does not result in a conflict. Lastly, request C has to be planned at 00:00-01:00, which will result in one constraint violation.

With look-ahead: 1 constraint violation. Project request A will first be scheduled. Every start time is tried, starting at 00:00 has the lowest cost both before and after look-ahead. The look-ahead shows that this decision would result in one constraint violation. Starting request A at 01:00 or at 02:00 will have higher cost. In both cases, request B will be greedily scheduled at 00:00, since this results in lower passenger hindrance costs. This means that request C will cause a constraint violation in both look-aheads. Therefore, request A will be scheduled at 00:00-01:00, each look-ahead shows a conflict and this has the lowest cost. Request B can then only be planned at 01:00-02:00 without a conflict. Finally, request C has to be planned at 00:00-01:00, so one conflict constraint will be broken.

The complete look-ahead addition is described in algorithm 6.1. Doing look-aheads to quantify the impact of the current decision on other requests strongly increases the required runtime. Therefore, only the most promising T starting times are considered and only a limited number of requests (N) are scheduled during that look-ahead. Some of the most promising times are just one day apart. In this case, it is not to be expected that a large difference can be found in a look-ahead. Therefore, all starting times which have a starting time with lower cost within 72 hours are removed from consideration for the look-ahead comparison. The different methods for determining which N requests are looked ahead to will first be explained. Then a few additional variations are explained. Finally, the results of a comparative analysis to determine the best configuration for the look-ahead are shown.

6.3.1 Determining related project requests

Since only a small subset of all requests can be considered in a look-ahead, it is important to determine which of the still unplanned requests are most influenced by the current request. In the look-ahead addition, a set of related project requests is therefore found for the current request. Then, these are sorted based on a priority function, and only the top T are scheduled as a part of the look-ahead. Multiple methods for both determining and prioritising the related project requests were considered. The goal of these methods is to find the requests which are most strongly influenced by the current decision.

When the required time window of the current request does not overlap with the required time window of the other request, they are considered to be unrelated, since in that case the current request is not expected to influence that future decision. Two requests which would cause a conflict when scheduled at the same time are considered related. In this case, the current request will restrict the available conflict-free scheduling periods of the other request, which could strongly influence that second decision. If at least one trajectory part is blocked by both requests, they are also considered related. In this case, they will often have the same best starting times. This makes the influence of the current decision on the cost incurred later larger.

Algorithm 6.1 Schedule a request greedily with look-ahead

```
1:  input: request $r$ to be scheduled in schedule $s$

2:  Initialise empty list of best starting times and their costs
3:  for time $t \in$ request's required time window do
4:      Move request to time $t$
5:      Add time and current schedule cost to list of best times and cost
6:  end for
7:  Sort list of starting times by ascending cost
8:  Remove similar starting times from list
9:  Remove all but the $T$ best starting times

10: Initialise variable for best starting time and best cost after look-ahead
11: for Remaining time in list of best starting times do
12:     Determine list of requests related to $r$
13:     Order related requests based on chosen priority metric
14:     Schedule $N$ first requests greedily without look-ahead
15:     if Current cost < best cost found so far after look-ahead then
16:         Update best cost and best starting time after look-ahead with current time
17:     end if
18: end for
19: Schedule request $r$ at best starting time after look-ahead
```

These rules for relatedness do not cover the full range of possible ways in which a current decision influences further decisions. For example, if the two requests have a lot of the same required essential personnel type, they also influence each other's scheduling freedom. However, these constraints were not as prevalent in the schedules for 2024, and often required different scheduling decisions to be made for multiple requests. Since the two earlier mentioned rules often already resulted in more related requests than can be looked ahead to, it was decided not to include more rules.

Then, when the set of related requests has been determined, this set needs to be pared down. This is done by taking the top N requests according to some metric. Three options for this were implemented and compared.

The first is to only take the N related requests that are first up in the regular scheduling order, which will be short-handed as the 'first' method. The most important advantage of this method is that it results in look-aheads that are closest to the actual greedy scheduling that will happen after the current request is scheduled. There are still unrelated requests that will be scheduled between the requests from the look-ahead during the actual scheduling, but no related requests are skipped. A possible downside is that the requests that are later in the scheduling order, but more influenced by the current request, may provide more information in the look-ahead.

Therefore, an alternative was considered, which takes the N requests which have the highest relatedness score. This score is computed based on the request currently being scheduled and each possible request to include in the look-ahead. This score was increased by 1000 if the two requests were a part of the same cluster. This was done to include cluster handling in the look-ahead principle. The score was then increased with the number of possible conflicts between the two requests. If requests conflict in multiple different ways, this means they are influencing each other more. This influence is expected to still be there even if many others are scheduled in between. The score was increased by one if the two requests have at least one trajectory part in common. Requests at the same location can be desirable to schedule together, in which case the first decision influences the next. This prioritisation method will be short-handed as the method 'largest'.

The third and final considered method is short-handed as 'most', this method will prioritise requests that are related to many others. The reason for this method is that requests which are conflicting with a lot of requests are expected to be difficult to schedule without conflict. Therefore, if such a request is looked ahead to more, it may make it easier to keep some starting times available for this request. The ordering is thus done based on the highest number of requests it is considered related to.

6.3.2 Additional look-ahead configurations

Besides different methods to determine the set of requests to include in the look-ahead, two additional variations on the basic look-ahead strategy were implemented.

6.3.2.1 Recursive look-aheads

An important weakness of the look-ahead strategy can be seen in example 2. When a conflict can be prevented by scheduling two or more requests in a suboptimal location, the look-ahead might not show this possibility. During the look-ahead, all requests are planned greedily. Therefore, this is a shallow look-ahead, that will not incorporate how future requests may also be influenced by their look-ahead. A possible, simple to implement, method to resolve this would be to use recursive look-aheads. Each of the requests in the look-aheads is planned using a look-ahead. This can theoretically be done to any depth. However, this results in an exponential increase in required computational time. In this thesis, we therefore only considered one level deeper. These recursive look-aheads would already take very long for a single full-year schedule, but it gives some insight in whether there are many of these constraint violations that can be resolved when the look-aheads are one level deeper.

6.3.2.2 Constraint-only look-aheads

One of the motivations for trying out a look-ahead strategy was to see if these look-aheads could prevent constraint violations, similar to example 1. For this reduction in the number of broken constraints, it is unnecessary to compute the full costs. Therefore, a constraint-only look-ahead was implemented. When this setting is used, the availability costs are not computed during the look-aheads. The look-ahead will therefore focus mainly on reduction of the constraint violations. This will reduce the time required for a look-ahead.

6.3.3 Hyperparameter analysis for look-ahead addition

To find the best possible configuration for the look-ahead, a small comparative experiment was done. These experiments were all done on the 2024-Q4 sub-problem. This problem was chosen, since it it is a lot faster, which makes it feasible to do some replications for these different configurations. It has a higher number of constraint violations than the full year problem, so it is suitable for seeing the ability to reduce this constraint violations.

The three prioritisation methods were compared. Based on these results, one method was chosen for each of the other experiments. Then, a full-factorial experiment with three different levels for the number of start times (T) is considered, using values 3, 5 and 25. The number of requests in a look-ahead (N) was included with two levels, 10 and 50. Besides, the recursive version was tested once. In this recursive version, the requests were scheduled with a look-ahead, one level deeper. Since the runtime for this was very large and the results were not very promising, no further schedules were created with this. The faster constraint-only variation was also experimented with. One setting combined with the most promising values of T and N was done, as well as one setting with the largest levels. Each parameter setting was repeated three times. All mean results are reported in table 6.2.

As can be seen in table 6.2, no large differences are visible between the three different prioritisation algorithms. Results were within one standard deviation of each other. Therefore, it was chosen to continue with the prioritisation that used the first requests in the greedy schedule order. The main reason for this is that further research will attempt to improve this greedy ordering (see chapter 7), these improvements could then also improve the look-ahead results further.

The results of the full-factorial algorithm show that the lowest level for the number of start times has the lowest cost results. This can be explained by the fact that if only the three best greedy options are considered, there are only limited cost incurring choices that can be made for reducing the number of violations. These costs are all lower than the best of five runs without look-aheads. This shows that this extra knowledge on requests being scheduled later is useful, and allows making better decisions. However, the extra computational efforts are quite large, so the trade-off is steep. Considering five starting times instead of three starting times has slightly higher costs, but lower number of broken constraints. Comparing 25 starting times does not clearly improve the results, but does influence the runtime markedly. Most likely, these starting times are almost never chosen.

In the amount of requests that should be looked ahead to, it can be seen that the lower level almost always performs better. This was explained by looking into the choices that were different from the

T	N	Priority	Additional	Runtime (hh:mm)	Cost	Constraint violations
0	0	n.a.		**00:15**	410.69	13.0
5	10	First		01:59	407.30	**9.3**
5	10	Most		01:41	407.10	10.3
5	10	Largest		01:44	406.20	**9.3**
3	10	First		01:13	**403.77**	13.7
3	50	First		02:20	404.57	11.3
5	10	First		01:59	407.30	**9.3**
5	50	First		03:29	413.12	10.3
25	10	First		03:18	405.85	12.0
25	50	First		05:27	408.60	10.0
3	10	First	Recursive	18:46	404.33	11.7
5	10	First	Constraint-only	00:50	409.86	13.7
25	50	First	Constraint-only	02:31	417.45	9.7

Table 6.2: Hyperparameter analysis on the 2024-Q4 problem, to compare different settings and techniques of the greedy constructive algorithm with the addition of a look-ahead. T is the amount of starting times that are compared with a look-ahead, N is the maximum length of this look-ahead. The prioritising method determines the order of the look-ahead requests. The best result in each column is boldfaced. Three repetitions were done with each configuration.

greedy. In the case of the longer look-aheads, these decisions are more often based on something happening further away in the scheduling. In some cases, this had different behaviour than what would actually happen in the remainder of the scheduling. This happened because there are no look-aheads within the look-ahead, and because the look-ahead schedules only part of the requests of the actual algorithm. Adapting your choice to expected future behaviour that is too far from reality made the decisions worse on average.

In conclusion, the five best starting times with a look-ahead of ten request will be used for the search strategy comparison. The look-ahead requests will be chosen based on their location in the scheduling order. However, from these three repetitions, no confident conclusions can be drawn, as the results are close. It is also not certain that this will outperform the regular greedy algorithm. It is expected that it has more potential on a full year schedule, since this will more often have to make decisions between multiple starting times that have a similar objective value on the partial solution, but influence future requests. The hyperparameters may not be optimised perfectly, as the look-ahead length will be relatively shorter compared to all requests, when using the full year.

6.4 Results

In table 6.3, the best and average results obtained by all solving methods for the railway maintenance scheduling are presented.

The hybrid greedy-evolutionary strategy found the best results, and was especially better at reducing the broken constraints. However, there was still quite a large variance in the number of hard constraint violations. Often, when more violations were present from the beginning of the second phase, only a few could be fixed. The cost of the hybrid is also significantly lower than the greedy algorithm.

The hybrid process per stage is visualised in appendix B.2. In the first stage, around 474 cost was generated, without any constraint violations. The first part of the improvement compared to the greedy algorithm comes from choosing the best of the 10 random greedy runs. This best individual had on average around 5 cost less than the worst starting individual. Afterwards, an average further improvement of around 22 was found in the first evolutionary search phase. Finishing the best individual greedily with the remaining request increased the cost by another 502, and on average 6 broken constraints. Lastly, the search when the schedule is full, caused a small cost increase of 1.5, with a mean decrease of the amount of broken constraints of 3.5. Solving these constraint violations cost more than

Search strategy	Runtime (hh:mm)		Mean solution quality		Best solution quality	
	Mean	Stddev	Cost	Constraint violations	Cost	Constraint violations
Greedy	**01:32**	00:01	1109.28	5.6	1106.17	3.0
Greedy with look-ahead	26:42	00:54	1084.98	8.2	1077.99	5.0
Hybrid	23:18	00:44	**1089.41**	**3.4**	**1091.13**	**1.0**

Table 6.3: Five repetitions of the most promising search strategies. The best value is boldfaced in every column. The best solution quality was found by the hybrid greedy-evolutionary strategy, which especially does best in reducing the hard constraint violations.

cost improvements could decrease. From this approximate behaviour, it can be seen that by doing the evolutionary search early, a lot of progress is possible. The mutations at the final full schedule are also useful, but for fixing some constraint violations, without a large cost decrease. Providing the current hybrid with even more runtime will most likely not give large further cost improvements. At the end of both evolutionary search phases, the improvements were smaller and further apart.

The look-ahead strategy did not prevent broken constraints in the way that was hoped. Many violations require either a change not found in the limited starting times or a larger amount of preventive changes which are not found by the shallow look-aheads. The constraint violations are higher than the greedy algorithm, due to suboptimal handling of the clusters. The cost is however consistently lower than the regular greedy. This shows that there is useful knowledge extracted from the look-ahead. However, the hybrid algorithm finds better results in a similar runtime. The look-ahead on average finds lower cost solutions than the hybrid algorithm, though more experimentation is required to establish whether these cost improvements would hold when the constraint violations are resolved.

The evolutionary algorithm, as expected, did not perform competitively. Two runs were done for 48 hours, the results of this are also visualised in appendix B.1. The best random starting individual has over 1,000 broken constraints. These were at first resolved fast, incurring high costs. After only 200 generations, the amount of hard constraint violations were reduced to only around 600 with a large increase in the costs. After that point, the hard constraint violations are dropping increasingly slow, but more cost improvements are also happening. After the first thousand generations, the solution still had more than 200 broken constraints. The cost at this point was still around 190 higher than the worst greedy solution found. After 48 hours, the cost was still around 95 higher than the average greedy solution found. This shows the progress slowing down, and the expected amount of iterations required being too high for the evolutionary to be a useful algorithm by itself. The convergence is different it was working for the year of 2019 and 2020, with the older problem definition. Back then, the cost is first further improved in a range lower than the final solution, before finally using the cooling to force resolving the violations, incurring some cost. However, the reduction of the number of constraint violations was too slow and the cost increase too high, to achieve a similar convergence with constraint cooling, also due to the smaller number of available generations.

6.5 Conclusion

To conclude, the two strategies that were able to provide a nice trade-off on the older problem definition and the years 2019 and 2020, are still performing well on this updated problem. The look-ahead addition shows a bit of cost reduction, however the constraint violations are on average higher. Therefore, the results are not competitive with the hybrid, which uses a similar amount of runtime.

Using these results, we are able to answer the research question “How do different search strategies solve the scheduling problem in terms of solving time and solution quality?”. Different search strategies can be a good method to provide a better trade-off between runtime and solution quality to the schedulers. The hybrid finds the best solution, whereas greedy finds worse schedules, but faster. Adding a look-ahead does not add significant value to this trade-off at this point. By further adapting the hyper-parameters for the hybrid, it is expected that the trade-off can be extended with possibilities between the fast greedy and the slower hybrid algorithm.

6.5.1 Future work

Based on the results of the researched search strategies, some potential future research into better solving of the maintenance scheduling problem is identified.

Greedy constructive algorithm. A few potential improvements to the greedy constructive algorithm were identified in this research.

The first and foremost one is to improve the order function used to determine the scheduling order. In chapter 7, this improvement will be further motivated and some results on this will be presented.

Besides, an important difference was seen between the greedy and the evolutionary algorithm. In the greedy algorithm, the schedule is created by trying out each day in the available time period. However, the same starting time is used for each of these. Evolutionary search is not limited by the granularity of the tried starting times. In some cases, being able to use different times of day makes it easier to fit requests which have lengths not an exact multiple of 24 in a good location. Some future work could therefore be done by trying the greedy with a smaller step size or some commonly successful times of day found in the hybrid algorithm.

Another identified possible research direction is an algorithm overarching the greedy. This could, similarly to the hybrid, do multiple randomised runs. Based on the intermediate results, it could filter and finish only the most promising ones. This is expected to be faster than the hybrid, but may already help in finding the best of multiple random runs with less runtime. By including a destruction heuristic, some of the techniques that were seen in the literature for iterated greedy algorithms could also be applied [53].

Look-ahead based greedy algorithm. Two main potential future research directions are identified for the look-ahead based algorithm.

The first is a rather simple change to the algorithm, which is strongly recommended. The greedy with look-ahead does not use the improved cluster handling from section 4.4.1. The reason for this was that the related requests always contained the cluster's other requests, so it was deemed less necessary. However, the broken constraints were at times still related to the shorter requests in a cluster. In some cases, the top T considered start times did not include any time suitable for these shorter requests. Therefore, including the shorter requests from the cluster in the original scheduling, instead of as a part of the look-ahead, is an important potential improvement.

Secondly, it may be beneficial to use less static hyperparameter settings. By making both the amount of start times and the length of the look-ahead more dynamic, the computational efforts can be spent more focused. Especially since the look-ahead mainly seems to be successful in reducing the costs. For example, if the top 10 start times are all very close, doing a look-ahead with more start times would be more likely to result in a cost reduction. In some situations, the first starting time is clearly better than the other. Doing look-aheads is not as likely to result in a different assigned starting time. Besides, doing look-aheads seems to provide more cost benefit in the beginning. Shortening the maximum look-ahead length and number of starting times towards the end could therefore reduce the runtime with only a limited decrease in solution quality.

Hybrid greedy-evolutionary algorithm. In the third search strategy, a good deal of future research could be done.

The mutations used by the hybrid algorithm are limited in its ability to solve certain constraint violations. More specialised mutations designed specifically to fix a certain constraint type could improve solving. Similarly, specialised mutations to decrease the costs further could be experimented with, either by solving soft constraints or by moving requests with higher availability costs in some heuristic method. Instead of designing new mutations, it may also be possible to achieve this to some extent with the current mutation. Research could be done into steering the mutations more towards important requests and adapting the probabilities of selecting certain types of request based on their success. For example, shift mutations could be more focussed on requests causing high cost other constraint violations.

A variance in the number of constraint violations was discovered in the results of the hybrid algorithm. The main reason was that the amount of constraints being broken in the second greedy phase of the scheduling differentiated quite a bit. Therefore, doing this part of the hybrid multiple times with some randomness may give the final evolution stage a better starting point, reducing this variance. This could be as simple as greedily scheduling the rest of the requests for all individuals in the population, or something more complicated could be investigated that immediately does some evolutionary search or restarts when constraints are broken during this second greedy phase.

In general, only very limited hyperparameter comparison was done for the hybrid algorithm, so this could also still be an interesting future research direction.

Adding look-aheads to hybrid. Ultimately, if improvements to the look-ahead strategy could make it faster and reduce the variance, we believe that this addition to the greedy may also provide improved solution quality in the hybrid greedy-evolutionary. The look-ahead gets lower cost than any of the greedy runs. Therefore, using this, especially for this first phase, but possibly also in the second greedy phase could potentially find even better solutions. This will of course increase the runtime even further, but for some phases within the creation of a schedule, this would be worth it if the solution quality is improved. Further research is necessary, especially to see if the cost reduction of the look-ahead still holds if the violations have to be resolved at a later stage. In general, breaking hard constraints allows more freedom to get low costs. Also, the benefit from the look-ahead and from the mutations in the evolutionary stage of the hybrid may have some overlap. If a better starting population is used, it needs to be researched whether this could still be improved as much by the evolutionary search. The hybrid has more freedom in assigning different times of day, and the look-ahead finds better cost solutions. This supports the possibility that there may be something to be gained from combining the two.

7 Schedule Order Optimisation

The performance of the greedy constructive algorithm strongly depends on finding the right order to construct the solution. All promising algorithms for solving the maintenance scheduling problem of ProRail use this greedy heuristic. In this chapter, the research question “How can new prioritisation techniques be applied to improve the solution quality?” is answered. This is done by first providing some necessary background and a more detailed motivation for focusing on this aspect of the algorithm in section 7.1. Then, a new order function design is proposed in section 7.2. This order function has weights that needs to be optimised, so in section 7.3 a strategy for this is proposed. Finally, the results of applying this proposed optimisation strategy, and the resulting optimised order function are given in section 7.4. Based on these results, the research question will be answered.

7.1 Motivation and background

7.1.1 Motivation

Some previous results support researching the schedule order. Adding randomness to the schedule order resulted in a large variance for both the cost and the number of constraint violations. This was seen both in some small additional experiment done now, and in the randomness experiments done for the year 2020 [5]. Some of these randomised order results were better than using the deterministic greedy plan order. Another motivation for this research direction is that positive results are applicable to each of the promising search strategies discussed in chapter 6. The performance of the greedy constructive algorithm is strongly influenced by having a good schedule order. This is a common property of greedy constructive algorithms [72]. Besides that, spending some effort to find a good order function may be beneficial as opposed to using this computational time in the search, since this order function can be re-used for multiple scheduling runs.

	Order	Cost	Constraint violations
2024	default	1109.69	5
	manual	**1107.97**	**0**
	random	1140.28	11
2024-Q4	default	408.80	14
	manual	**403.73**	**0**
	random	419.21	35

Table 7.1: A single greedy deterministic run using the default order function, a manually created order, and a completely random order for two different input problems: one full year (2024) and one created sub-problem of the fourth quarter (2024-Q4). The best result is boldfaced for both problems.

A manual order was created to evaluate the potential of improving the solution quality by finding a better greedy schedule order. Requests that were regularly involved in constraint violations and requests with large variable costs were moved forward in the order. This manual order creation was first done for the difficult 2024-Q4 sub-problem, and later also for the full year of 2024. The differences between using the default order function (eq. (4.1)), the manual order, and a completely random order are given in table 7.1. The completely random order shows the large impact that the order has on the performance; with a random order, substantially worse schedules are found, with more constraint violations and higher costs. With the manual order, all constraint violations could be resolved for both problems without a decrease in costs.

7.1.2 Background

In this section, some background for the techniques used in this chapter is given. Some related work with the usage of an order function is given first. Then, the background on the simulated annealing that is used to optimize the weight in section 7.3 is given. Finally, some background concerning conflict-guided methods, which are similar to the mutation used in the simulated annealing, is provided.

Priority-based algorithms. (Incremental) priority algorithms is an alternative name used for the group of algorithms that are greedy, where the order of the additions to the partial solution is determined based on some priority metric [73]. The greedy algorithm in this thesis falls within this definition of priority algorithms. This priority can either be fixed or adaptive, a fixed priority means that the priority is determined only on the properties of the possible next additions to the constructed solution. An adaptive priority also includes information of the current partial solution to determine the priority [73].

Priority metrics are also often used in online scheduling, where priority rules are required to determine the schedule for jobs that arrive stochastically [74]. Priority algorithms have also successfully been applied to numerous other problems, such as subset-sum [72].

Simulated annealing. In 1983, simulated annealing was introduced by Kirkpatrick, Gelatt Jr, and Vecchi [75]. It is a nature-inspired metaheuristic for optimisation. For a predefined number of iterations, a heuristic is used to find a neighbouring state of the current solution. If the update improves the quality, it is always accepted. If the quality decreases, it is accepted with a probability based on the current temperature and the amount of quality decrease. Every iteration, the temperature decreases and the algorithm becomes increasingly less likely to accept worse quality solutions. The most important advantage of this method is that it can be generically applied to many problems, without tuning a lot of hyperparameters. It is especially well suited to problems with “either numerous, contradictory constraints, or [a] complex, baroque cost function” [76]. Only one solution is kept at a time, which also makes it memory-efficient in comparison to population-based methods.

Conflict-guided methods. In this chapter, the principle of trying to learn from where failure occurs is used. This principle can be seen in numerous algorithmic techniques. An example is in conflict-guided search in SAT solvers, where a conflict clause is learned to speed up search. Another example of learning from conflicts is given by Boussemart, Hemery, Lecoutre, *et al.* [77], where constraints are weighted based on the amount of time it caused a failure. The variables are then ordered based on the sum of all weights of this variable’s constraints. This principle was later extended by Balafoutis and Stergiou [78] with ageing and a weighting based on the amount of domain reduction the constraint caused, rather than only the number of conflicts.

7.2 Designing order function

It is preferred that the order is determined with a method that can be used for multiple input problems. The function can then be experimented with on a smaller, comparable (sub-)problem. This allows optimising the function with more iterations. Besides, when new requests become available, the existing function is also suitable for this updated problem. Optimising the order for every change to the requests, or configuration variables, is not desirable. These changes might often happen during certain stages of the scheduling process. Besides, being able to try out the impact of these changes quickly makes it easier to identify how realistic this request is, and how much it would influence the schedule.

A simple approach was decided on, which may be limited in its ability to find the optimal order. This decision was made, since this function will by design be less prone to overfitting to a specific input problem. Besides that, explainability of the results is better for simpler functions. However, every aspect of difficulty can impossibly be captured by this simplistic function. The main goal of this is to see if the result improvement is possible with a simple improved order, using some domain knowledge. Better order functions could in that case be researched using this as a starting point.

As such, a simple linearly weighted function was created with seven input features. This function was then used as an importance score for each of the requests that had to be scheduled. The requests are thus ordered from high importance score to low importance score, before greedily adding them to the schedule one-by-one.

7.2.1 Input features

Domain knowledge was used for determining features that are important in determining the priority of a request. Generally, the most difficult or most important requests should be scheduled first. For example, when a request conflicts with many other requests, it may not have any starting time available when it is scheduled later. There are only a limited number of weekends in a year. In the weekend, there are fewer passengers, meaning that maintenance work scheduled in the weekend will have lower availability costs. Having a request which hinders a lot of passengers early in the planning order increases the likelihood that there is a suitable weekend available for this request.

All the input features are explained one-by-one in this section. These input features do not cover every property of a request. Some experimentation was done with a few other features. However, with the following seven features, enough dimensions were present to prioritise the requests most often resulting in constraint violations for the 2024 input set. Besides, the largest aspects of the expected variable costs are covered, so other cost-related features are not expected to make a large difference.

Length. The length of the request is an important attribute influencing the priority of a request. Longer requests should generally be scheduled earlier. If a long request is scheduled later, the probability of there being a starting time available without any conflicts is lower. Besides, when a request needs to be planned at a suboptimal starting point, the expected extra incurred cost is larger for a longer request.

Personnel costs. The personnel costs are multiplied with a factor based on the scheduled time. For example, personnel is more expensive at night than during the day, and Sundays are more expensive than other days. Besides that, if a short request can be scheduled together with another request at the same trajectory part, the costs do not have to be scaled up to at least eight hours. The requests with the largest personnel costs with limited passenger hinder would ideally be scheduled in these limited, cheaper periods. In order to achieve this, these need to be scheduled earlier.

Conflicts. Another attribute that influences how difficult it is to correctly schedule a request, is how many potential conflicts there are. Therefore, the total sum of other requests, preplanned hinder and hard constraint level events with which the request conflicts forms the third input feature.

Essential personnel. The amount of required essential personnel can be important for preventing broken constraints. When only requests with low personnel demands are early in the plan order, these will generally be spread out over the year to improve on other aspects of the objective. This could then leave no available spots for requests with high essential personnel. The project request attribute for this is computed as the sum of the three types of required essential personnel. Currently, the limits of the three types of essential personnel are comparable, if these grow further apart, these numbers should be weighted relative to the limit.

Time window size. The size of the time window shows how much freedom there is in finding a starting time for the project request. A small time window is more restrictive for longer requests than shorter requests. Therefore, the time window size relative to the project request length is added to the importance function. This attribute is thus defined as the length of the request divided by the time window size.

Passenger hinder. An important part of the variable costs is directly related to the number of blocked passengers. Only a limited amount of time is available when less passenger travel than regular, such as holidays. Therefore, it would be preferred to use this space for the requests which hinder most passengers, and they should have a higher importance. Therefore, the sixth attribute is the number of blocked passengers at the default hour by the request.

Goods hinder. Another part of variable costs is related to the goods hinder. Similarly to the passenger hinder, scheduling requests with a high hindrance of freight travel earlier could decrease the total cost. So, the number of freight streams requiring trajectory parts affected by the project request is the last feature.

7.2.1.1 Feature normalisation

Normalising the features gives them a comparable value range. This makes optimisation of function weights better, as mutations are more comparable. Furthermore, it makes the results easier to interpret. To do this, a simple mapping of all features to a [0,1]-range and a z-standard normalisation were implemented. Based on the initial results, the simple mapping to the unit interval was used with a mean-based starting point.

The first, simple option for normalising the input features is to map all values to the range between 0 and 1. In this case, each value was divided by the maximum value in the set of input project requests, except for time window size, which is already between 0 and 1. When using this method, the distribution could still be different within this interval. If all weights of the order function are equal, this does not mean that all attributes are equally influential. Therefore, an equal weight increase will have more impact on the scheduling order for one feature than for another.

To reduce this, it was decided to use it with a mean-based starting point. Each of the weights was determined such that the mean value of each attribute multiplied with that weight is equal. This already resulted in a better score as a starting point, and also made it find more diverse orders and better quality early in the optimisation.

Besides, a z-standard normalisation was implemented. In this normalisation, the mean of the attribute values is subtracted from all attribute variables, and they are divided by the standard deviation [79]. Contrary to the expectations, this did not improve upon the initial results. Therefore, this standardisation was not used in further experiments. This might be because the values with a larger variance were more suitable for smaller changes. However, it could also well be that the mutations were just better tuned to the first normalisation.

7.2.1.2 Adding interaction with length

Many of these features are expected to have a strong interaction with the length of the request. A request which conflicts with many others would for example be of higher importance, as it would be more difficult to have a non-conflicting spot for it. However, if the request is very small, it does not need to be as early in the scheduling to still have good starting time available.

Because of this, a small attempt was done to see whether multiplying some of these properties with the length of the request would improve results. The length multiplication seemed to influence the attribute value too much, and it became too similar to a length-based order. Besides, the difference in difficulty between for example a 24-hour and a 36-hour maintenance request is not actually that large.

Therefore, an alternative for this length interaction was created, where the length was mapped to a multiplication factor as follows:

$$\text{length multiplication} = \begin{cases} 0.4 & \text{if length} \leq 8\,, \\ 0.7 & \text{if length} \leq 48\,, \\ 0.9 & \text{if length} \leq 168\,, \\ 1.0 & \text{else.} \end{cases}$$

Requests that can be performed in one night are valued lowest. Requests that can be scheduled within a weekend are somewhat higher. Everything above a week is valued again a little higher. These multiplication factors slightly improved the quality of the resulting order function in initial results and were therefore used in further experiments.

7.3 Optimising order function weights

Now that an order function has been designed, the next step is to optimise the weights to find the importance of these different attributes. The relevant problem characteristics and the chosen optimisation strategy will first be explained. Then, some problem-specific mutations that were developed will be explained. Finally, the results of these weight optimisations will be given and analysed.

7.3.1 Optimisation strategy

Many numerical optimisation methods could be applied to optimise the weights of the order function. One of the important properties of the problem is that back-propagation to update weights is difficult, since we do not know the perfect order. We only know the score that indirectly results from this order function. A gradient descent based technique is therefore not expected to be very successful for this problem. Small changes to a certain weight may barely impact the resulting weight. It may for example only cause one small change at an unimportant position, that reduces the quality slightly. Whereas a larger change to that same weight could move more requests in the resulting order and improve the quality.

Furthermore, an optimisation strategy with the option to include problem-specific mutations to the weights would be preferred. In the process of manually reordering the requests, it could be seen that including knowledge on which requests cause large increases in variable costs or broken constraints might increase the speed of finding good weights.

Based on these aspects, combined with the background as explained in section 7.1, it was decided to use simulated annealing as the optimisation strategy. This is a simple strategy, expected to be good for an expensive objective. Besides, it works well with continuous variables, and allows for adding custom mutations. The individuals for the weight optimisation are not very memory efficient and can not be parallelised well, so the fact that simulated annealing is not a population-based method is also preferred.

The simulated annealing starts with a single individual, which is a list of weights for the order function. These are evaluated by applying the function with these weights to the requests, and using the resulting order to create a schedule with the greedy algorithm. The quality of this resulting schedule is used as the fitness for the optimisation. For n generations, a mutation is done which updates those weights, by sampling a new weight from a normal distribution centred at the current weight. The standard deviation is decreased based on the current temperature. During the beginning of the run, larger mutations can be useful for good exploration. When the temperature is decreased, smaller steps are preferred, to better converge to the best local point. The new weights are evaluated and compared to the current fitness. The temperature is updated after every generation, $T_{i+1} = \alpha \cdot T_i$.

To determine if the new weights are accepted, a few different situations are relevant. If the fitness improves, the proposed weight is always accepted. If the current fitness has no constraint violation and the proposed weights do result in violations, a separate acceptance probability is used, since this moves from feasible space to infeasible space. This acceptance probability is configured for the start and the end, and exponentially decreases over iterations. If the fitness gets worse, but the feasibility of the current and the proposed individual are equal, the mutation is accepted with a probability depending on the current temperature and the amount of decrease in quality as given in eq. (7.1). If the number of constraint violations increases, this is most important. A constraint violation is scaled to the same magnitude as the costs. If the number of constraint violations is equal, the cost difference is used for determining the acceptance probability

$$p_{accept} = e^{-\frac{\Delta_{accept}}{T}} \tag{7.1}$$

$$\Delta_{accept} = \begin{cases} \Delta\texttt{constraint violations} & \text{if } \Delta\,\texttt{constraint violations} > 0 \\ \Delta\texttt{costs} & \text{else} \end{cases} \tag{7.2}$$

If no improvements are found for a certain number of iterations, the entire search process is restarted. The best solution found in all iterations and restarts is stored, to use as the final solution.

7.3.2 Constraint-guided mutation

In order to find good weights as fast as possible, it is desirable to steer the search towards good regions. A conflict-based approach for this was considered. Using knowledge from where the computation runs into problems can help to find a good solution faster. Which requests caused a constraint violation is stored. Specifically, the request that was scheduled in the greedy construction step where the violation occurred is considered. For violations involving multiple requests, the earlier scheduled request that it conflicts with is thus not used for the constraint-guided mutation. The assumption behind this mutation is that if the constraint-violating request had been scheduled earlier, there may have been a starting time available that did not cause any violations. This is supported by the success of using this when manually reordering to find a good schedule order, as explained in section 7.1.1.

This mutation was at first done by increasing the weights of the importance function proportionally with the attributes of all requests causing a constraint violation. However, it was quickly seen that this did not always have the desired effect of moving this request forward. At times, there were more other requests with higher values for the highest value attribute, while a lower value attribute had fewer higher ranked requests. Therefore, the relative value of the feature was used instead of the absolute value. The weight increase was done in proportion to the position the request would have in the order if only that specific attribute was used for sorting. With the set of conflict causing requests $P_{conflict}$ and the index of request x when sorting only using attribute a as $I_{x,a}$. A weight for the constraint-guided mutation strength is defined as w_{cg}. This weight is linearly decreased for every constraint violation later in the original schedule order. The first constraint violation is hardest to resolve by moving it forward in the ordering. It has the least available weight changes that will move it forward. So, by giving this update the most weight, this request will have better probability of being moved forward, since its changes will not immediately be undone by the later constraint violations. The complete formula of the constraint guided mutation is therefore:

$$w_{i,x+1} = \begin{cases} w_{i,x} + \sum\limits_{p_k \in P_{conflict}} \dfrac{I_{p_k} - I_{p_k,i}}{I_{p_k}} \cdot w_{cg} \cdot \dfrac{k}{|P_{conflict}|} \cdot \dfrac{T}{T_{start}} & \text{if } I_{p_k} > I_{p_k,i} \\ w_{i,x} & \text{else} \end{cases}$$

In this formula, it can be seen that the constraint-guided mutations are also weighted based on the temperature. The constraint-guided mutation is done as an addition to a regular, normally distributed mutation, to maintain some randomness. The standard deviation of the regular mutation is divided by two when a constraint guided mutation has already been done.

To further demonstrate the desired function of the constraint-guided mutation, example 3 is used. This shows how the weight values of properties that could increase the request's importance value are increased.

Example 3. Current weights: $w_{length} = 0.9$, $w_{conflictingness} = 0.8$, $w_{cg} = 1.0$
Order:

- Project request A: length 50, conflictingness 100
- Project request B: length 100, conflictingness 50, constraint violation

First, constraint-guided, project request B causes violation:
$w_{conflictingness} = 0.8 + \frac{2-1}{2} = 1.3$
$w_{length} = 0.9$
Then, normal mutation with half the standard deviation:
$w_{conflictingness} = 1.3 + \mathcal{N}(0, 0.5) = 1.18$
$w_{length} = 0.9 + \mathcal{N}(0, 0.5) = 0.87$

7.3.3 Cost-guided mutation

A mutation was designed with the goal of decreasing the costs, similar to the constraint-guided mutation. A preprocessing step is added, which schedules each request in an empty year schedule. The number of variable costs this incurs is stored. This will also be called the reference cost. After the current weight function is tested, we can check how much more expensive the current schedule choice is in comparison to that original incurred cost. So, for example, a request could cause a cost increase of 10 if the schedule were empty, whereas it increases the total variable costs with 20 when scheduled at the end. The requests with the largest difference between this current incurred cost and the reference cost are attempted to be moved forward in the planning order. For this 'moving forward', the same principle was used as in the constraint-guided mutation. The five requests with the highest relative variable cost increase were used in this mutation.

However, looking into the effects of this mutation, it was seen that this did not result in the desired cost decreases. One important possible cause was found for this. Some of the requests that cause high relative cost increases can not be scheduled earlier without some constraint violation happening later. Or there is some other, conflicting, project request that has a larger cost benefit from using the starting time used to obtain the reference cost. So, attempting to get closer to achieving this reference cost does not improve solution quality, since that would cause more detriment to other requests than gain to this request. One method was implemented to counteract this. After every iteration, the reference cost r_c was updated with the current increased variable cost c_i as follows:

$$r_c = \alpha \cdot r_c + (1 - \alpha) \cdot c_i$$

Different values were tried for α. This adaptation did not improve the results of the cost-guided mutation. It either took a long time to stop trying to move requests forward that could not decrease the cost, or the relative cost differences converged to zero and the requests chosen to be moved forward were almost random.

Some possible future research directions into better mutations are worked out in section 7.5.1. However, for this thesis, it was chosen to focus on different aspects of improving the order function optimisation. The cost-guided mutation is thus not used in the optimisation of the weights for the order function.

7.3.4 Comparable sub-problem

When the weight optimisation is run for the full year, each evaluation takes at least an hour. This means that only a limited number of weights can be tried. This makes optimisation difficult. Therefore, it was decided to use a smaller, comparable problem. The weights can be optimised based on this smaller problem, and then applied to better solve the full year problem.

Weights that perform well on this sub-problem are expected to also perform well on the full problem, if the smaller problem is representative. This means that the smaller problem is comparable in which parts of the cost can be optimised most and which constraints are difficult to satisfy. Finding this representative sub-problem is a challenge, and might not be equally possible for every input data set. For the year 2024, it was chosen to use six sub-problems. How these sub-problems were created is explained in section 3.3.3, and some more characteristics are given in appendix A.

Each of these sub-problems was solved with the greedy constructive algorithm. The results for this with the data set of 2024 can be found in table 7.2. From this, it can be seen that the requests are quite evenly spread over the periods. The most hours, highest costs, and most constraint violations can be found in period 5. Since the full year also had some constraint violations, this period was chosen as it was expected to be the most comparable. This sub-problem was used to tune the hyperparameters and design good mutations for the simulated annealing. It is used to show how much improvement could be obtained, first on this sub-problem and then also by applying these weights to the order function for the full year. It was also used for trying the new order function in combination with different search strategies. At a later stage, the best-performing hyperparameters and mutations were then also applied to each of the other sub-problems, to also be able to evaluate the importance of the chosen sub-problem.

Period	Number of requests	Schedule hours	Cost	Constraint violations	Runtime (mm:ss)
1	132	10 680	158.71	0.0	11:07
2	139	8904	190.68	0.0	13:06
3	138	8928	162.71	0.0	11:45
4	88	7200	192.20	1.0	08:49
5	131	11 040	241.03	9.0	12:31
6	100	7704	186.96	0.0	09:01

Table 7.2: Characteristics of the six sub-problems of 2024, and the result of solving with the greedy scheduling algorithm.

7.4 Results

In this section, the results of all experiments to identify the performance of using the proposed order function are given. No runtimes were measured for these experiments, as the order function does not have a large impact on the runtime compared to the same search strategy with the regular order.

7.4.1 Hyperparameter analysis for order function weight optimisation

To find the right hyperparameters for the simulated annealing process, a small comparison was done. Due to the limited effectiveness of the simulated annealing independent of the hyperparameters, the decision was made not to do a complete comparison.

First, a few small tests were done to find the most important configurations, and the approximate range of values for these hyperparameters to use. The starting temperature was set at ten million. A hard constraint violation was valued as five million per violation. The cost-guided mutations were not used, and the constraint guided mutations were done with a probability of 0.5, and a weight of 0.05. There were two parameters left to compare. The first is the standard deviation of the mutations (σ), which was included with levels 0.025 and 0.05. Second, the decrease of the temperature (α) was included with the levels 0.95 and 0.975. For all four combinations, seven repetitions with the 2024-P5 problem were done. It can be seen that the mean based starting point already starts at only one constraint violation. The results with the smallest mutations were generally the worst, so an extra configuration was included with a standard deviation of 0.1.

The results of one experiment per configurations can be seen in fig. 7.1. The other experiments are left out of the figure for clarity. The behaviour of the algorithm was equally inconsistent in the different runs. The improvement during the optimisation process is not very large and there is no good convergence visible. When temperature drops too far, even in very suboptimal sport, no more changes are accepted. After the restart period of thirty iterations without improvement, the process restarts. This happens quite often in the process, as can be seen by some longer stretches of suboptimal solutions. The lines do not converge nicely, it seems to almost be a random search. The search space might be too irregular for this optimisation method. This is seen from the fact that improvements are found at random moments with larger jumps. Almost no exploitation is done when a better quality solution is found.

However, the total number of constraint violations and cost is decreasing a bit, and the starting point is already an improvement compared to the original order function. Therefore, the slightly better performing hyperparameters with the larger mutation size and a higher temperature (α=0.95 and σ=0.1) are used from here on. In order to improve the weights of this linear order function, a more consistent optimisation is necessary. For this work, the decision was made to focus on evaluating the potential of using a new order function.

7.4.2 Order function weight optimisation

The simulated annealing was done for 200 iterations for each of the six sub-problems. This took on average around 25 hours per sub-problem. The results of this can be found in table 7.3. All sub-periods have a better solution quality than they had with the original order. Part of the improvement results from the newly designed function independent of the weights. A bit more profit was found during the weight optimisation.

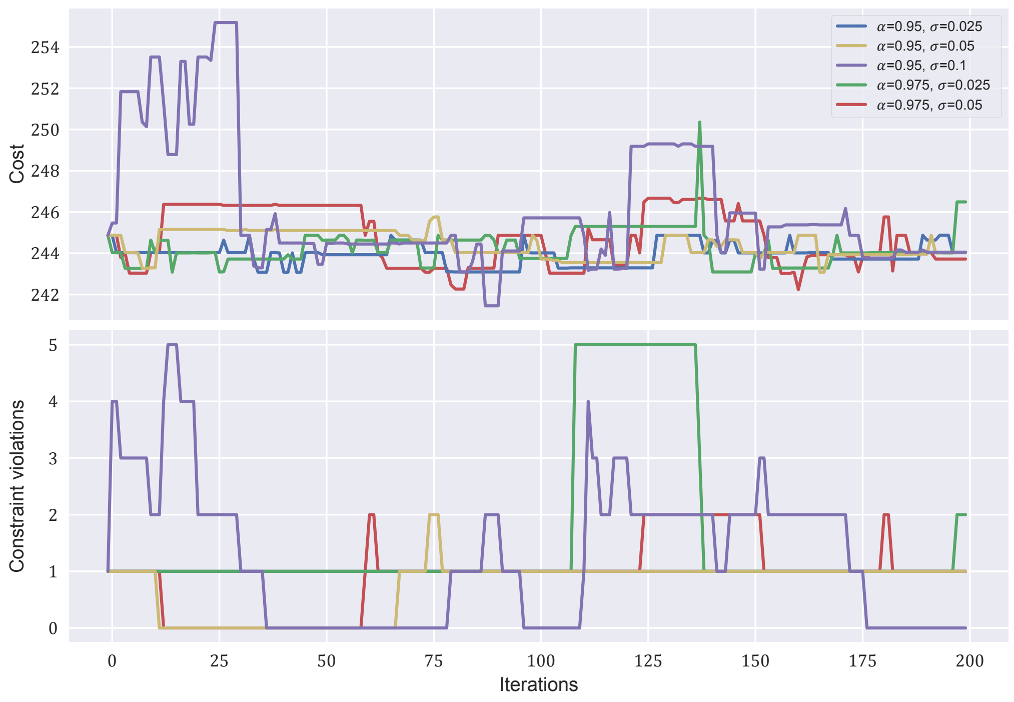


Figure 7.1: Convergence plot of different hyperparameter settings. One run of the simulated annealing per configuration for optimising the weights of the order function for 2024-P5 is shown for clarity. Seven replications were done for each hyperparameter setting.

The optimised weights are quite a bit apart. Which shows that either the sub-problems require very different weight values, or the process of finding good weights is not very consistent. Based on the hyperparameter analysis, this second reason is most likely. Weights that are quite different could still have almost equal resulting solution quality. The starting point of the simulated annealing is dependent on the distribution of the attribute in the sub-problem, so each weight optimisation starts at a different point. This spread is also still seen in the final found weights.

Problem used to optimise	Optimised weights							Solution quality	
	w_{ess_per}	w_{pass}	w_{good}	w_{confl}	w_{window}	w_{len}	w_{reg_pers}	Cost	Constraint violations
2024-P1	1.073	0.126	0.492	0.279	3.640	0.666	0.724	156.58	0.0
2024-P2	0.050	1.015	0.784	0.190	4.837	0.045	0.079	181.81	0.0
2024-P3	0.605	0.052	0.300	0.300	4.274	0.432	1.036	155.02	0.0
2024-P4	1.573	0.223	0.372	0.144	3.837	0.167	0.685	184.43	0.0
2024-P5	0.918	0.473	0.503	0.290	3.023	1.296	0.497	244.81	1.0
2024-P6	0.678	1.388	0.245	0.164	3.100	0.476	0.949	180.31	0.0

Table 7.3: The solution quality and weights resulting from the simulated annealing to optimise the weights of the order function after 200 iterations, on each of the six sub-problems.

7.4.3 Applying optimised order function with different search strategies

To see the impact of the new order on finding a complete annual schedule, the new order function is applied to the year of 2024. The function with the weights optimised on the 2024-P5 problem was applied with the search strategies from chapter 6. The results of this experiment are displayed in table 7.4.

Search strategy	Mean solution quality		Best solution quality	
	Cost	Constraint violations	Cost	Constraint violations
Greedy	1110.86	0.0	1106.83	0.0
Greedy with look-ahead	1087.50	1.3	1087.64	1.0
Hybrid	**1089.81**	**0.0**	**1084.42**	**0.0**

Table 7.4: The three most promising search strategies with the new order function, using the weights optimised using the sub-problem 2024-P5. The highest quality mean and best solutions are boldfaced.

With all strategies, better schedules were created with the new order function as opposed to the default order function. The number of broken constraints is always lower. The fast, greedy algorithm already finds a better schedule when a good order is used, than any solution found with the default order for the 2024 dataset.

The greedy search strategy obtains a very similar cost. This is especially good, since satisfying constraints often incurs a cost. The hybrid generates the best quality results, as it did with the default order function. The cost of the look-ahead is again slightly lower, but this still has one broken constraint. This constraint violation seems to be the result of the limitations in cluster handling.

The difference between hybrid and greedy is smaller with the new order function. Both solve the problem without any constraint violations. The hybrid greedy-evolutionary algorithm has a higher cost than with the default order. This smaller difference with greedy is to be expected; the starting point is already better, so finding improvements in the evolutionary search is more difficult. When no constraint violations are present, the space to make mutations is more restricted. With a few broken constraints, mutations that resolve a constraint violation and break a different one are possible.

7.4.4 Comparison of weights from different sub-problems for order function

To determine the robustness of the results, and to see the importance of which sub-problem is chosen to optimise the weights, all the optimised weights were tested with the greedy algorithm on the full year. The results of this are given in table 7.5.

Order	Mean solution quality		Best solution quality	
	Cost	Constraint violations	Cost	Constraint violations
default	1109.28	5.6	1106.17	3.0
2024-P1	1112.86	0.0	1109.78	0.0
2024-P2	**1101.35**	**0.0**	**1096.69**	**0.0**
2024-P3	1113.04	0.0	1112.03	0.0
2024-P4	1110.98	0.0	1109.20	0.0
2024-P5	1110.86	0.0	1106.83	0.0
2024-P6	1106.75	0.4	1106.42	0.0

Table 7.5: The mean and best solution of five repetitions of the greedy algorithm, solving the 2024 problem with different order functions. The default is the original order function, the new order function is the proposed function with the different weights optimised on the different sub-problems. These weights can be found in table 7.3. The lowest cost mean and best solution is boldfaced.

It can be seen that the order function performs better than the default order function with all optimised weights. Almost all runs have no broken constraints. The weights optimised on the 2024-P2 sub-problem result in the best final solution quality. As can be seen in table 7.3, these weights are also a bit more extreme. The reason for this is that many changes to the order function weights had only small impact on the quality of this sub-problem. This caused the simulated annealing to get further away from the starting point, where a local optimum was found. This optimum had more cost focused weights, which resulted in better cost scores. This shows that the order function may be further improved with a better weight optimisation, that could also find these weights on different sub-problems.

From the fact that it always improves compared to the default, it can be concluded that the precise weight values are not as important as the design of the order function. The sub-problems are generally comparable enough to optimise the weights to obtain a function that reduces the constraint violations consistently, often to zero, without large increases in cost. If better weights are found, even more improvement is possible

7.5 Conclusion

Improving the scheduling order is a powerful tool to obtain better results, which can improve every successful search strategy defined in the previous chapter. None of the search strategies was able to resolve all constraint violations with the default order function. When problematic requests arrive late during the scheduling order, it is extremely difficult to still plan these feasibly. It often requires moving many requests to a different starting time. The hybrid algorithm therefore already does a first round of mutations before the schedule is too full. This gives the ability to solve constraint violations (or higher cost) before the freedom to move requests is too restricted by a full schedule. However, even with the hybrid algorithm, the project requests that are being scheduled only in the second phase can still be very difficult to plan feasibly or with a low cost. Moving multiple requests in the correct way is often impossible to achieve when the schedule is already quite full. Designing mutations that are better at satisfying the constraints requires more problem-specific algorithm design. With the limited amount of generations that are currently used, the quality of the greedy starting population becomes more important. A good order function is shown to be a good way to improve these starting individuals.

This leads us to an answer to the research question considered in this chapter: “How can new prioritisation techniques be applied to improve the solution quality?” Applying a new order function shows a lot of potential for improving the solution quality. The computational time required for finding the order function does not necessarily have to be updated for every change to the input data or configuration. This is very desirable, since the scheduling team can get better results in the more orienting phase, despite the desired fast solving. This way, they can better see the impact of changes to the configuration or fixing the starting time of certain requests. Since this prioritisation is used in all techniques, it improves results of each search strategy. The look-ahead improves relatively more, since it struggled most with constraint violations.

Since each input data set may be different, an ideal solution to this scheduling order problem would be to find some prioritisation or ordering method that is reusable, but can also be optimised to handle different areas of difficulty. For example, if a certain year has many constraint violations caused by high essential personnel, the ordering method needs to be adaptable to this. Or, if the conflict constraints cause most violations, requests which are more likely to be creating conflicts need to be scheduled earlier.

In this chapter, we have proposed a possible function for improving the order function, which significantly improves the solution quality of the maintenance schedules for 2024. The most important advantage of this method is the fact that it is a function that can easily be applied to different input problems. On the 2024 schedule it performs very well and results in some strong improvements to the constraints with limited increase in costs. Good weight optimisation is not that essential for its performance. However, with the right weights the greedy algorithm can even result in lower cost solutions without constraint violations.

There are a few drawbacks to the proposed solution. The first drawback is that to keep the feature dimensionality low, some domain knowledge based features were created. These do not cover all possible constraints, and may therefore not be as functional on different input years. Besides that, the proposed weight optimisation does not consistently perform well. Quite different, optimised weight values perform similarly. The function design more strongly impacts the good results than the weight optimisation. The constraint-guided mutation helps to guide the search a bit, but besides that, it is almost exclusively exploration and no exploitation.

7.5.1 Future work

Based on the results from the presented order function research, some future work is identified. Possible improvements to different aspects of the proposed method are given. Then, some possible future research into new techniques related to the priority of the requests is identified.

Design of order function. The current input features do not cover all possible types of constraints. Adding more features could allow finding more complex orders, which also take these constraint types into account. Furthermore, interactions between the different features could be further researched. Some attributes may only be relevant when combined with others. In the current function, only the interaction with the length is taken into account. An adaptive order function could be another interesting improvement. Including the current state of the schedule into determining which request needs to be scheduled next could provide information that could improve the order. For example, requests which have a larger part of their available time window already filled with other requests should be scheduled earlier.

Optimisation of order function. From the performance of the proposed method, it can be seen that changing the schedule order can reduce both cost and constraint violations. The latter is easier to obtain, since reordering based on where the constraint violations occur is a very simple and effective method. To find orders that reduce the cost, it was more difficult to learn from the performance of the current order. Without this ability to learn directly from the current order, improving this is more dependent on finding a comparable sub-problem. More direct learning or different cost-related attributes are aspects that could be investigated more. Besides, not just looking at moving requests earlier, but also at different aspects that define a good schedule order, could be an interesting future research direction. For example, for some requests, scheduling them at the same time can reduce the costs. If these are immediately after each other in the schedule order, the probability that the second one can still be feasibly scheduled at the same spot is higher. These are some possible directions to consider in developing new mutations in the function optimisation. Doing research into completely different optimisation methods could also be interesting.

Other techniques. The most important future research direction would be to look into other techniques for determining the order. An importance function can maybe not completely capture the difficulty of determining a good order for a problem with such a large amount of conflicting goals, with all constraints and different aspects of the cost function. Priority rules, or ensembles of priority rules, such as used in online scheduling, would also be a way to allow capturing more information on the difficulty as compared to the current linear attribute weighting. These rules could be used for comparing a pair of requests, which can then be applied to find an order for the complete set of requests that follows these pairwise decisions as well as possible.

Besides that, directly learning an order, rather than a function, may provide better results. This would require spending more computational resources for finding these good orders, and this would not be as reusable. However, the results in this chapter show that this might actually be worth the additional runtime, due to the large positive impact on the solution quality that is obtained with a simple function already. It would also make reducing the constraint violations simpler, as the concept behind the constraint-guided mutation could then be applied easily.

It may also be possible to learn the order function completely. Some previous work on using genetic programming for priority rules was done [80]. However, the computational requirements of evaluating a function are quite high. Besides, the resulting score does not provide direct feedback to which aspects of the current function do or do not function well. Based on the results from this chapter, it would therefore be difficult, and the other future research seems more promising. However, if successful, it could provide new insights on the problem and result in more reusable functions that can be retrained on new problem input. Learning a pairwise comparison could be interesting to consider when using genetic programming to learn an order. Comparing two requests and determining which one should be greedily added to the schedule first can potentially be more easily labelled for supervised learning.

A different possibility would be to add a manual priority. Some requests may be found more important by the scheduling team or the contractor than others. Incorporating this into the scheduling order would maybe help in finding good orders. Besides, it would improve the quality of the solution for the end users, independent of the corresponding cost value.

8 Generalisation to Different Input Years

The research from the previous chapters provided more insight into the trade-off between runtime and solution quality that can be obtained in the solving of the yearly maintenance scheduling of 2024. However, for these techniques to be useful on the long-term, it is important that they have a similar effect in different trade-offs of the problem input of different years. The last research sub-question: "How do techniques that improve the available trade-off between the runtime and solution quality for the schedule of 2024 generalise to different years of input data?" will be answered in this chapter. This will be done by applying the different techniques to the years of 2023 and 2025. In section 8.1, the detour approximation techniques are used. The different search strategies are compared in section 8.2, and in section 8.3 the order function is applied to the different years. The properties of these input problems were given in section 3.3.2.

8.1 Detour approximation techniques

The detour approximation techniques are the first techniques that were applied to different input years. The decision was made to leave out the average delay strategy here. This technique was dominated by other strategies on the 2024 input year, as it had both worse runtime and worse solution quality. The results are given separately for 2023 and 2025. The main conclusion from these results is that these techniques are also able to reduce runtime with negligible impact on the solution quality for different input years. The impact is smaller for these problems as they have less passenger hinder.

8.1.1 2023

The results of applying the detour path approximation techniques to the input year 2023 are given in table 8.1.

Detour approximation technique	Cost		Constraint violations		Runtime	
	Mean	Stddev	Mean	Stddev	Mean (hh:mm:ss)	Stddev (mm:ss)
ADWN	472.75	3.34	**5.0**	0.0	00:44:20	00:21
ADWNSymm	474.09	1.99	**5.0**	0.0	00:37:15	00:18
Exact	**471.90**	1.53	5.4	0.6	01:02:06	03:05
ExactSymm	473.58	0.95	5.6	0.6	00:48:32	03:20
TNR	472.31	3.55	5.4	0.6	00:39:24	00:20
TNRSymm	474.38	1.55	**5.0**	0.0	**00:33:51**	00:07

Table 8.1: Five repetitions of a greedy scheduling run for the input year of 2023, with different detour path approximation strategies. The best result for each column is boldfaced. ADWN is the improved version of the baseline. TNR is transit node routing. The Symm techniques uses the symmetrical version of the passenger streams. A detailed explanations of these techniques is given in section 5.3.

The greedy algorithm solves the maintenance schedule for 2023 more than three times as fast as the 2024 schedule. Using an exact path calculation, the year of 2023 is solved in around an hour and 2024 takes over three hours. This is explained by the fact that the year of 2023 contains fewer project requests, with on average shorter length. The project requests also block less trajectory parts. Since less detour paths need to be computed, the runtime saved by using a detour path approximation strategy is also less than it was in 2024.

However, similar reductions in runtime are seen for the schedule of 2023 as well as the schedule of 2024. The symmetrical strategies are again faster than the non-symmetrical versions. The exact strategy is slowest. The transit node routing with symmetrical path merging is the fastest strategy and saves almost 50% of computation time, as it also did for the year 2024.

The solution quality is very close for all of these strategies. Compared to the 2024 input year, the difference between the symmetrical and non-symmetrical is larger. However, all cost differences are still within one standard deviation.

8.1.2 2025

The results of applying the detour path approximation techniques to the input year 2025 can be found in table 8.2.

Detour approximation technique	Cost		Constraint violations		Runtime	
	Mean	Stddev	Mean	Stddev	Mean (hh:mm)	Stddev (mm:ss)
ADWN	563.36	1.19	0.0	0.0	01:20:46	00:01:13
ADWNSymm	564.09	2.13	0.0	0.0	01:01:27	00:01:09
Exact	562.79	0.61	0.0	0.0	02:03:23	00:02:45
ExactSymm	**562.45**	1.14	0.0	0.0	01:29:23	00:01:58
TNR	564.41	3.36	0.0	0.0	01:08:16	00:01:22
TNRSymm	564.44	3.71	0.0	0.0	**00:55:56**	00:00:41

Table 8.2: Five repetitions of a greedy scheduling run for the input data set of 2025, with different detour path approximation strategies. The lowest cost and runtime is boldfaced, constraint violations were always zero. ADWN is the improved version of the baseline. TNR is transit node routing. The Symm techniques uses the symmetrical version of the passenger streams. A detailed explanations of these techniques is given in section 5.3.

The computational time required for solving the 2025 input year is between the 2023 and 2024 greedy solving times. The strategies obtain a speed-up with negligible cost differences, as it also did for the years of 2023 and 2024. By using detour path approximation, the runtime can be reduced with more than 50% compared to the exact computation. In five replications, the randomness added to the greedy algorithm has more impact on the solution quality than the chosen technique.

8.2 Search strategy comparison

The different search strategies that performed best on the 2024 dataset were the greedy algorithm and the hybrid greedy-evolutionary algorithm. It is important to see if this performance can also be achieved on different problem inputs. The look-ahead addition is also experimented with in this chapter, since the results were close to the hybrid technique for the year 2024. Besides, it may provide some more insight into the strengths and drawbacks of this method and aid potential future research into (parts of) this technique.

8.2.1 2023

The results of the different search strategies on the 2023 input problem are given in table 8.3. In comparison to the 2024 results of the different search strategies, a few differences were discovered.

Search strategy	Runtime (hh:mm)		Mean solution quality		Best solution quality	
	Mean	Stddev	Cost	Constraint violations	Cost	Constraint violations
Greedy	**00:37**	<00:01	474.09	5.0	471.65	5.0
Greedy with look-ahead	08:31	00:17	**449.61**	**2.3**	**447.46**	**1.0**
Hybrid	12:31	00:47	462.71	3.3	460.75	3.0

Table 8.3: Five repetitions of the greedy algorithm and three repetitions of the greedy algorithm with look-aheads and the hybrid greedy-evolutionary algorithm with the input for the year 2023. The best result in each column is boldfaced.

The runtime is always lower than it was for the 2024 data set. This is caused by having fewer and less hindering requests. The distribution of the runtime over the different methods is also not the same. When running the techniques with the 2024 maintenance, the look-ahead addition was around 10% slower than the hybrid greedy-evolutionary algorithm. Using the same hyperparameter settings, the look-ahead addition is substantially faster than the hybrid for the 2023 dataset. The main reason for that is the fact that the 2023 requests have fewer related requests than the 2024 requests. Because of this, the look-ahead is more often shorter than the configured cut-off.

Besides, there are some other differences in performance of the search strategies, concerning the solution quality. The greedy algorithm still gets the fastest results, and has worse solution quality compared to the more elaborate search strategies. However, the greedy algorithm with the additional look-ahead strategy obtains the best schedules for the 2023 input, whereas it had more constraint violations than the hybrid for the 2024 year. The cost difference between these two strategies is also larger in 2023. The hybrid has a similar reduction in both cost and constraint violations as the 2024 input. The main difference is that the look-ahead addition performs better using the 2023 data. The first explanation is the lack of clusters in this input year, with which the look-ahead addition has problems. This may not fully explain the larger cost difference yet. To be more confident in why the look-ahead works better here, more experiments are required. It may be due to the fact that there are generally less related requests, which cause more focused look-aheads. If, for example, a request has only 20 conflicting requests, taking 10 of these into account is more likely to yield a better decision, than when there are over 80 conflicting requests. The fact that constraint violations were reduced compared to the greedy algorithm suggests that more violations could be prevented with these look-aheads for 2023.

8.2.2 2025

The results comparing the runtime and solution quality of the different search strategies can be found in table 8.4.

Search strategy	Runtime (hh:mm)		Mean solution quality		Best solution quality	
	Mean	Stddev	Cost	Constraint violations	Cost	Constraint violations
Greedy	**01:01**	00:01	564.09	**0.0**	560.71	0.0
Greedy with look-ahead	07:39	00:04	560.19	34.7	563.18	23.0
Hybrid	14:19	01:34	**555.82**	0.3	**555.84**	**0.0**

Table 8.4: Five repetitions of the greedy algorithm and three repetitions of the greedy algorithm with the look-ahead and the hybrid greedy-evolutionary strategy with the input for the year 2025. The best result of each column is highlighted.

From these results, it could immediately be concluded that the 2025 input was not suitable for the look-ahead addition, as this year contains many clusters. The greedy algorithm with the look-aheads does not include all requests from the cluster to determine which times need to be compared with a look-ahead. Therefore, only one replication of this was done. The runtime of this version was lower than the hybrid due to the smaller number of related requests, similar to the 2023 input data.

The greedy algorithm already consistently solves the problem without constraint violations, which shows that this input year is less difficult to feasibly solve. The only remaining objective in this case is to reduce the cost. The cost of the hybrid is around 8.5 lower, which is relatively less than both 2023 and 2025. This problem has more freedom in scheduling available, due to less restriction from the hard constraints. Most maintenance work could already be scheduled in low hinder hours with the greedy algorithm, so moving requests can not reduce the hinder as much. This may also mean that the amount of generations could be lower for the hybrid version with 2025, without losing as much solution quality.

8.3 Schedule order model

In chapter 7, results showed that changing the order function could reduce the constraint violations in schedules created for the 2024 maintenance with only a small cost increase. To see if this proposed order function can be usefully applied to different years, this method for creating an order function will also be applied to the input years 2023 and 2025. Besides gaining insight in the advantages and drawbacks of this proposed order function, it will also show if similar improvements in quality are possible on different problems. First, the weights optimised for 2023 and 2025 are given, and the results obtained with these weights combined with different strategies. Then, the comparison is made between reusing the weights that performed well on 2024, as opposed to optimising these weights specifically for this year. Finally, the results of a small additional experiment are presented, which further analysed the potential of changing the order to reduce the number of constraint violations.

8.3.1 Optimising weights specifically for new year

For both 2023 and 2025, the same process was applied to generate representative sub-problems as for 2024, as explained in section 7.3.4. However, since both these years have a smaller number of requests, it was decided to use quarterly periods, instead of two-month periods, since this would be a more comparable problem size. For 2023, the third quarter was used for optimisation, as this had the highest cost and two constraint violations in a greedy scheduling run. For 2025, the fourth quarter was used, which contained the most project requests and incurred the highest cost using the greedy algorithm. The weights that were found with simulated annealing are given in table 8.5, which also includes the 2024 weights used for the result of the year 2024 for comparison.

Used problem	Optimised weights						
	w_{pers}	w_{pass}	w_{good}	w_{confl}	w_{window}	w_{len}	w_{PEAT}
2023-Q3	0.848	1.137	0.835	0.798	1.671	0.492	1.219
2024-P5	0.918	0.473	0.503	0.290	3.023	1.296	0.497
2025-Q4	1.229	0.395	0.613	0.382	3.183	0.641	0.556

Table 8.5: The weights found using simulated annealing to optimise the weights of the order function with 200 iterations, for 2023, 2024, and 2025.

The simulated annealing process resulted in the most different weights with the 2023-Q3 sub-problem. This set of requests had different mean attribute values, which determine the starting point. Besides, that, the algorithm eventually found a local optimum, where the violations of the sub-problem were reduced from 2 to 1, which was further away from the starting point. The constraint guided mutations also had a different effect here, as the requests most often breaking a constraint had different properties.

During the simulated annealing process for the 2025 sub-problem, both the constraint violations and around 2.5% of the cost were reduced compared to the starting point. Especially the essential personnel weight was increased due to those constraint violations. These constraint violations were not found on this sub-problem with the default order, but were there with the mean-based starting point for the 2025 simulated annealing.

8.3.2 Performance in combination with different search strategies

The results of the comparison can be found in table 8.6, which also includes the default order results for comparison. The results from the year 2024 are also included to provide a clear overview of the effects of the order function on all years. From these results, a few observations are highlighted.

Input year	Search strategy	Default order		New order function	
		Cost	Constraint violations	Cost	Constraint violations
2023	Greedy	474.09	5.0	474.11	8.2
	Greedy with look-ahead	449.61	2.3	452.85	3.3
	Hybrid	462.71	3.3	**452.89**	**2.0**
2024	Greedy	1109.28	5.6	1110.86	0.0
	Greedy with look-ahead	1084.98	8.2	1090.54	1.0
	Hybrid	1089.41	3.4	**1089.81**	**0.0**
2025	Greedy	564.09	0.0	560.53	0.0
	Greedy with look-ahead	560.19	34.7	563.45	26.7
	Hybrid	555.82	0.3	**557.11**	**0.0**

Table 8.6: Overview of all three input years with the most promising search strategies, both with the default order function and the new proposed order function. The new proposed order functions use the optimised weights as explained in section 8.3.1. For 2024, five replication of all algorithms were done. Five greedy replications were done on the 2023 and 2025 datasets, and three hybrid greedy-evolutionary replications and the greedy algorithm with look-ahead addition. The highest quality mean and best results are boldfaced for all three years.

The proposed order function performs best on the 2024 input data, which shows that the design of the function and the optimisation process was possibly too focussed on this specific problem. Changing the weights to the specific properties of the input year did not result in equally large solution quality improvements for other years as it did for 2024.

From the greedy algorithm results, it can be seen that 2023 greedy algorithm finds worse schedules. The cost and constraint violations of a greedy scheduling run per newly scheduled request is visualised in fig. 8.1. It can be seen that the objective of the constraint-guided mutations is achieved well for the 2023 scheduling, as all violating requests are early in the schedule with the new order function. However, this did not reduce the violations as was expected. There are multiple potential causes for this, and to better explain these results, a small additional experiment was done, which is explained in section 8.3.4. The greedy algorithm does find better schedules with the new order for the year 2025, No constraint violations could be reduced, but a nice cost reduction was found.

The look-ahead addition improves with the new order function on the 2024 and 2025 input, but in both cases, it does not result in schedules as good as the hybrid strategy. For the 2023 year, the look-ahead addition was actually the best technique with the default order. However, this algorithm performs worse with the new order function.

The hybrid solving method is improved when the order function moves more cost and constraint breaking requests towards the beginning of the greedy scheduling process, into the first phase. The proposed order function is able to do this moving forward well for both 2023 and 2024, as can be seen in fig. 8.1. With the hybrid strategy, this resolves all violations for 2024 and some for 2023, while improving the cost in the 2023 case. For the 2025 case, the cost is not moved forward as much. This may explain the fact that the hybrid greed-evolutionary did not reduce the costs equally well as for 2023 and 2024. The mean constraint violations did improve. With the default order, one of the three repetitions had a constraint violation, and this did not occur with the new order. The run with the constraint violation also had the lowest cost. This influences the mean cost of the hybrid with the default order.

From these results, it can be concluded that the performance of the proposed order function depends both on the used search strategy, and on the properties of the input data. A method for determining the right order should ideally incorporate both of these factors into the analysis.

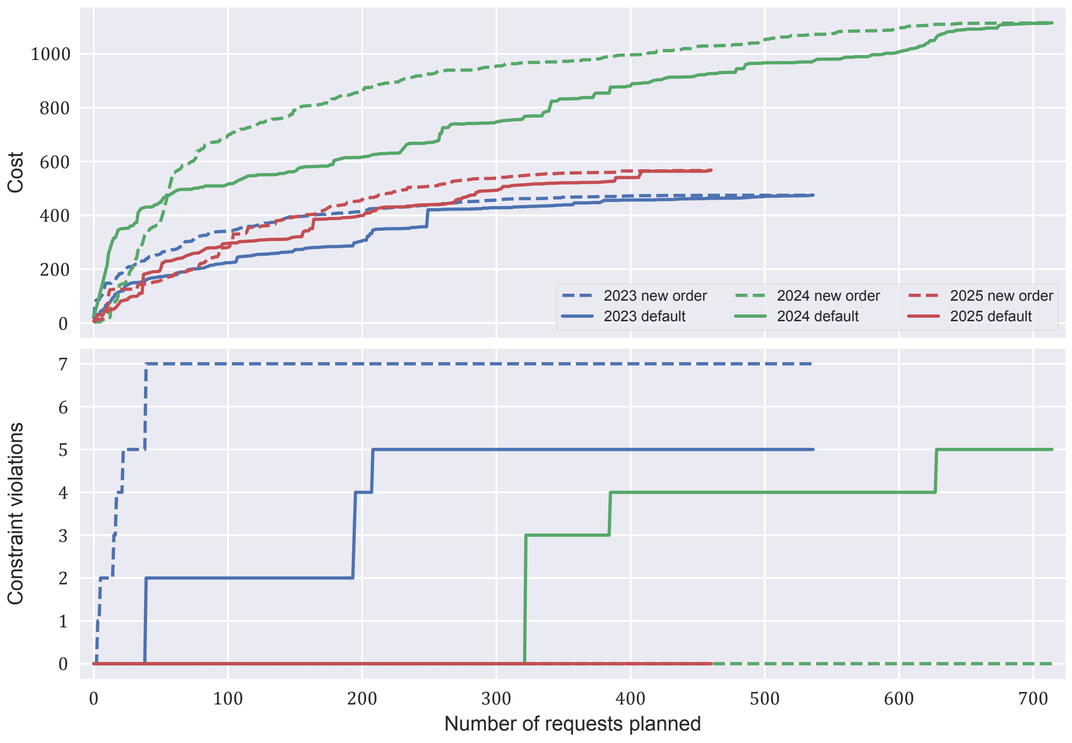


Figure 8.1: The cost and constraint violations during a greedy scheduling run, for all input years, with both the default order and the new order functions.

8.3.3 Re-using weights from 2024

In order to see how well the weights that were optimised for the year 2024 perform for different input years, some of the weights are also applied to the other input years. The results of this are given in table 8.7. It was done both for the weights that were used for most experiments, as they were optimised based on the most comparable sub-problem: 2024-P5. The weights optimised based on 2024-P2, which had the lowest cost greedy schedules for 2024, were also tested on different years.

This shows that for the 2023 input year, the greedy algorithm performs worse when using the new function with all the tried weights, compared to the default order function. The weights as optimised on 2024 perform better than the weights specifically optimised using a sub-problem for 2023. It is important to note that this performance on the greedy algorithm does not necessarily mean that it would also perform better with a hybrid algorithm. From previous results, it was seen that the demands of a good order are different depending on the used algorithm.

For 2025, the new order function with the weights optimised on 2024-P5 result in the lowest cost schedules being created. All order functions did not result in any constraint violations. The new order function with the weights optimised specifically for 2025 do perform better than the default order function, contrary to the 2023 results.

The 2024-P2 weights were also included in these results, as these weights were best at reducing the cost for the 2024 greedy algorithm. The weights are more extreme; they more strongly weigh some features. They were included here to see if these best performing weights for the greedy on 2024, were also best for the other years. Results show that these weights do not generalise as well as the 2024-P5 weights, and perform worse on both 2023 and 2025. The cost is also higher than the weights optimised for these years specifically.

So, it can be concluded that weights optimised on one input year can also perform well on another. However, the best-performing weights can not be determined independent of the problem.

Search strategy	Order function	Mean solution quality		Best solution quality	
		Cost	Constraint violations	Cost	Constraint violations
2023	default	**474.09**	**5.0**	**471.65**	**5.0**
	new with 2023 weights	474.12	8.2	472.75	7.0
	new with 2024-P2 weights	477.49	7.0	474.71	7.0
	new with 2024-P5 weights	472.47	6.6	471.72	6.0
2025	default	564.09	0.0	560.71	0.0
	new with 2025 weights	560.53	0.0	558.17	0.0
	new with 2024-P2 weights	564.87	0.0	563.45	0.0
	new with 2024-P5 weights	**559.11**	**0.0**	**557.14**	**0.0**

Table 8.7: The mean and best schedule quality for 5 replications of the greedy algorithm for different ordering function. The new order function and the 2024-P2 and 2024-P5 weights are explained in chapter 7. The 2024-P5 weights were used for most experiments, since this sub-problem was most similar to the full problem in the first analysis. The 2024-P2 weights performed best, out of the six tried weights with the greedy algorithm, on the 2024 input data. The 2023 and 2025 weights are explained in section 8.3.1. For both years, the best obtained solution qualities are boldfaced.

8.3.4 Good order for greedy algorithm for the year 2023

As was already briefly stated in section 8.3.1, some unexpected results were obtained when using the newly proposed order function in combination with the 2023 year. To fully explain these results, and understand the changes required to design an order method that can handle this input data set as well as the year 2024, some more research is required. A small start on this was done, which will be explained here. This experiment was not enough to completely explain the results, but helps provide more confidence in the strength of optimising the order to obtain better quality solution for the yearly maintenance scheduling problem in the Netherlands.

The proposed order function does not properly cover some required differentiations to find this right order for the 2023 input data. A few different reasons for the order function not being able to find a successful order were found.

First, 2023 had more defined preplanned hinder and dependencies than 2024 and 2025. These are included in the order function, but with an equal weight as other conflicts. So, a request that conflicts with another request in the input gets an equal value for the conflicts feature as a request that conflicts with a preplanned hinder. However, it may be easier to prevent conflicts with these other requests, as these are not fixed. Therefore, this is an aspect of difficulty that is not properly covered by the proposed order function.

Secondly, there were more instances of constraint violations in 2023 occurring due to too much concurrent work per project. Some projects in 2023 had a large number of requests. Since none of these were clustered, they all had to adhere to the concurrency constraints. This constraint is not covered in the order function.

Moving requests that are breaking constraints forward was possible to some extent, but did not reduce the number of constraint violations. It seems that the performance of the greedy is also more dependent on the specific order within these first, most difficult requests. Based on these missing aspects in the order function, it is hypothesised that an order function in general could improve greedy solving in 2023. However, the order function proposed in this thesis was unsuccessful. There it was attempted to validate this hypothesis, and show that the constraints are not actually too difficult to find a solution with fewer violations using a greedy algorithm with a better order. This was done using a very simple algorithm, that did many greedy iterations of the 2023 problem. After each iteration, the requests breaking a constraint were moved forward, to around halfway between the first request and its current position. Since the greedy algorithm for the year of 2023 was fast, it was possible to do quite a few iterations. After having tried around a hundred different orders, it found an order that only had 2 constraint violations with a greedy algorithm. This is significantly better than both the default order and the new order function. This shows that improvement of the greedy algorithm is possible with a better order, also for the year of 2023.

8.4 Conclusion

The research question that this chapter aimed to answer is: “How do techniques that improve the available trade-off between the runtime and solution quality for the schedule of 2024 generalise to different years of input data?”. The answer of course strongly depends on the used technique, as these can have different effect on the algorithm’s ability to cope with certain difficulties. Some obtained results were similar to the results of 2024. Other aspects of the research done had somewhat different outcomes when applied to a different year of maintenance scheduling input.

More efficient computation of the objective and detour path approximations, which have only very limited effect on the objective precision, were seen to be effective ways to speed up computation, which could be readily applied on different years. For both 2023 and 2025, substantial speed-ups were again achieved, with negligible reduction of solution quality.

The search strategies did not perform the same for all input problems, especially the greedy algorithm with a look-ahead addition turned out to be problem-dependent. For all input years, the greedy algorithm found decent quality results, and the hybrid greedy-evolutionary algorithm improved the solution quality using more runtime. Where the hybrid algorithm required slightly less runtime than the greedy with look-ahead for the year 2024, it required more for both 2023 and 2025. The greedy algorithm with look-ahead had the best solution quality for the schedule of 2023, but found substantially worse quality solutions than all other search strategies for the year 2025.

A priority function to determine the greedy scheduling order can improve an algorithm’s ability to cope with constraints with all input years. However, the requirements of this order are dependent on the characteristics of the problem and the used solving method. The new order function, as proposed in this thesis, was not equally successful on the different input years. Whether an order function is successful was also seen to be more strongly dependent on the used strategy in different years, while it improved the results with all search strategies for the year 2024. For example, the 2023 hybrid algorithm improved with the new order function, whereas the other two search strategies obtained worse schedules than with the default order. The proposed order function was not solid enough to consistently improve performance, but the potential to reduce constraint violations by finding a good order was seen in all used years. Small improvements are obtainable with a generally good order function, but the ability to optimise based on the problem input characteristics and used search strategy is necessary for larger improvements.

8.4.1 Future work

First and foremost, creating a larger set of representative input problems to test with would be an important future research direction. These input problems could be synthetic, or based on historic data. Having access to a quick assessment of how a proposed change to an algorithm would work could help to find more robust algorithms. This would require a set of problem inputs with different degrees of precise specifications, as well as different areas of the objective that are more complicated.

In this chapter, it was seen that the greedy algorithm with look-aheads performed very well on the 2023 input year. Better understanding of why this algorithm performed better here can aid understanding of this method, and how it can potentially be used in the future.

Improving the order for the greedy scheduling algorithm improved resulting solution quality for the different years as well. However, future research into better ways for adapting the order based on the problem input and the search strategy is necessary to find a more robust solution. This could improve solution quality, without additional runtime for every schedule generated. For example, it was seen that the order function, when used in combination with the hybrid algorithm, had different requirements. The best results were obtained when it moves the right requests into the first phase. This knowledge could be used to optimise directly based on only these first 100 requests if the order will be used for a hybrid algorithm. This would make order optimisation faster, and it could potentially make the resulting orders better in combination with the hybrid algorithm.

In the weight optimisation for the different input years, a mean-based starting point was used. However, the 2024 weights resulted in better schedules with the greedy algorithm. Therefore, some research into using the weights from a different year as a starting point for optimisation could be interesting.

9 Conclusion and Future Work

In this chapter, the main takeaways from this research are reiterated and the main research question is answered. Finally, the most important future work directions resulting from the work in this thesis are presented.

9.1 Conclusion

The conclusions and answers to the research questions defined in section 1.3 are shortly summarised. These answers are used to answer the main research question.

1. **How do different aspects of the problem and the currently used algorithms influence the runtime and solution quality?**
   The runtime and solution quality are negatively impacted by the high problem complexity. The large number of constraints and cost parts, many of which are specified on a very high level of precision, are the most important aspects giving this problem its complexity. These are not all linear, and many are conflicting. The high dimensionality is also complicating; a schedule contains hundreds of maintenance works. This makes finding good solution quality difficult. The current algorithms allow some speed-ups by storing the current state and only computing the impact of changes made to the schedule. However, the resulting algorithm complexity and the high memory demands are important drawbacks of this algorithm. A substantial improvement to the computation time was achieved by removing some implementation inefficiencies.

2. **How can approximation of passenger detour paths be used for faster solving?**
   Approximation of this part of the objective is an effective way to reduce runtime, with negligible impact on the solution quality. Especially the methods that find feasible alternative paths, which are not necessarily the optimal shortest path, performed well. With these methods, there is only limited error in the path length and in the resulting passenger availability costs. This has no detrimental effects to the decisions made by the greedy scheduling algorithm. Therefore, solutions of equal quality can be found. The fastest approximation developed is a transit node routing strategy. This approximation uses the structure of the network and the common transfer patterns in different paths to more efficiently compute a detour path. Even faster scheduling is possible by estimating the hindrance of passengers from station A to station B based on the detour paths from station B to station A.

3. **How do different search strategies solve the scheduling problem in terms of solving time and solution quality?**
   Different search strategies are a strong way to allow schedulers to choose between runtime and solution quality based on the demands of the current phase in the scheduling process. The hybrid greedy-evolutionary finds the best solution, whereas the greedy algorithm finds worse solutions, but substantially faster. A novel strategy was developed to search for good maintenance schedules, by adding a look-ahead to the greedy scheduling. These look-aheads allow finding lower cost solutions. However, due to a higher number of constraint violations, it does not add value to the trade-off at this point.

4. **How can new prioritisation techniques be applied to improve the solution quality?**
Applying a different order for greedily adding elements to a maintenance schedule shows great potential for improving the solution quality. A good order prioritises project requests that benefit most from having increased scheduling freedom. The success of a prioritisation was discovered to be dependent on the input data set characteristics. The computational time required for optimising the order only has to be spent once for the input year. Since the greedy heuristic is used in all algorithms, it can improve the solution of each search strategy. For the 2024 maintenance schedule, a simple order function based on domain knowledge can resolve all constraint violations with negligible incurred cost.

5. **How do techniques that improve the available trade-off between the runtime and solution quality for the schedule of 2024 generalise to different years of input data?**
On different input years, some techniques performed as well as they did in the 2024 schedule, others were more dependent on the specific problem input. Detour path approximations could be readily applied to different years. Runtime was reduced with negligible reduction of solution quality for the years 2023 and 2025. The optimal search strategy was problem-dependent; the greedy algorithm with a look-ahead addition performed better than hybrid for 2023, but substantially worse for 2025. A strong priority function to determine the greedy scheduling order can improve the ability to cope with difficult constraints for different input years. However, the requirements of this order is dependent on the characteristics of the problem and current solving method.

9.1.1 Answer to the main research question

The main research question to be answered in this thesis is: **"How can improving different problem and solution aspects of the yearly maintenance scheduling on the Dutch railway network help to obtain better understanding and improved usability of the trade-off between runtime and solution quality?"** This question can impossibly be answered in totality based on this research, due to the very broad formulation. In this thesis, a few directions for better understanding and improved solving are explored. With the combination of the improvements obtained in these directions, this research achieved faster solving and better solution quality. It gives schedulers the ability to once more trade off runtime and solution quality based on the current requirements in the scheduling process. All results are first summarised. Then, the main takeaways from this thesis are explained.

9.1.1.1 Overview of results

A compact overview of the results from all content chapters is found in table 9.1. The first row of this table displays the starting point of this thesis, which can be considered the baseline. However, the objective function was still slightly different here, as it used a path approximation for this final evaluation as well, and a small mistake was present in one of the constraint handlers. With the first improvements to the greedy algorithm and the objective calculation, which were explained in chapter 4, a first big reduction in the required runtime was made. The scheduling of clustered requests was also improved; this reduced the cost and constraint violations of the solutions found by the greedy algorithm for the years 2024 and 2025.

By applying detour path approximation techniques, the runtime was reduced further by around half. With these runtime improvements, more complex search strategies became feasible. A comparison was made between the greedy algorithm, a novel variation on the greedy algorithm with look-aheads, and a hybrid greedy-evolutionary. The results showed that it is possible to obtain better solutions, but it requires more runtime. The hybrid greedy-evolutionary algorithm resulted in the best schedules for the years 2024 and 2025. The greedy algorithm with look-ahead found better schedules for 2023.

Finally, a new order function was introduced that could be applied to each of these strategies. With this new order, all constraint violations could be resolved for 2024 and the solution quality of all search strategies was improved. The results of the proposed order function was not consistent for the different years, and was also seen to be more dependent on the used solving method. In conclusion, the new order combined with the hybrid greedy-evolutionary resulted in the best mean solution quality for all three considered years.

Used techniques			2024			Chapter 8					
						2023			2025		
Search strategy	Detour approx.	New order	Runtime (hh:mm)	Cost	Constraint violations	Runtime (hh:mm)	Cost	Constraint violations	Runtime (hh:mm)	Cost	Constraint violations
Unimproved greedy[1]			25:23	1181.1	22.0	14:32	476.5	12.0	10:27	600.9	24.0
Greedy			03:29	1107.5	4.8	01:02	471.9	5.4	02:03	562.8	0.0
Greedy	×		**01:32**	1109.3	5.6	**00:37**	474.1	5.0	**01:01**	564.1	0.0
Greedy with look-ahead	×		26:42	1085.0	8.2	08:31	449.6	2.3	07:39	560.2	34.7
Hybrid greedy-evolutionary	×		23:18	1089.4	3.4	12:32	462.7	3.3	14:19	555.8	0.3
Greedy	×	×		1110.9	0.0		474.1	8.2		560.5	0.0
Greedy with look-ahead	×	×		1087.5	1.3		452.9	3.3		563.5	26.7
Hybrid greedy-evolutionary	×	×		**1089.8**	**0.0**		**452.9**	**2.0**		**557.1**	**0.0**

Table 9.1: Compact overview of all results from this thesis. All 2024 results are the mean values of five repetitions. The 2023 and 2025 result with greedy algorithms are also the mean from five repetitions. The hybrid greedy-evolutionary and the greedy with look-ahead algorithm was done with three repetitions for 2023 and 2025. The differences between the unimproved and the regular greedy are explained in chapter 4. The different solving methods are all explained in chapter 6. The detour approximation is done using the symmetrical version of the improved baseline (ADWNSymm), as explained in chapter 5. The new order function is the order function optimised on a sub-problem from that year, as proposed in chapter 7.

[1]The final evaluation used slightly higher cost evaluation here, as sometimes suboptimal passenger detour paths were found, a small mistake was also present that sometimes resulted in inaccurate number of essential personnel constraint violations.

9.1.1.2 Main takeaways

Better understanding of the most important aspects of the problem was achieved throughout this process. This knowledge could help to improve the problem modelling now and in the future. The scheduling problem is still being updated with more detailed information. Potential inefficiencies in the problem model and how these can be prevented in the future were identified throughout this research. Two main examples are given here.

First, the high granularity in the passenger data was shown to increase the runtime unnecessarily. By merging certain origin-destination pairs, no solution quality was lost. Also, the fact that passenger data can be specified differently for every hour increases runtime. However, this addition only marginally increases the accuracy of the model.

The second example is the impact of the high number of unique precision levels used to define constraints. Removing some of these levels could help in reducing the high memory demands and speed up computation. When designing new constraints, it should be considered whether they can be defined on one of the existing levels rather than adding a new level.

Another important takeaway from this research is the two important methods that were discovered to handle the consequences of the more realistic objective; reducing the computational demands of the objective evaluation and improving the performance of the greedy algorithm.

Reducing the computational demands of the objective evaluation is one of the most important methods to speed up solving. More efficient evaluation is necessary to find decent schedules quickly, or better schedules with more available runtime. The main improvements to the objective evaluation runtime were achieved by focussing on the aspects of computation that were repeated most often. These most frequent calculations can be improved by spending more time on the implementation details, or applying approximation techniques.

The second method was to improve the greedy algorithm. It became more important to obtain good results with the simple heuristic algorithm. Metaheuristics generally require more objective evaluations and are therefore less effective with the more complex objective. The improved heuristic performance will in turn also improve algorithms using this heuristic as a subroutine. In this problem, for example, improving the greedy algorithm by finding a better schedule order was a good way to resolve difficult constraints.

The conclusions derived from this research provide direct improvements for this problem, however these conclusions can also be of interest for other use cases.

The results in this research can be useful for other countries scheduling their railway maintenance. Better understanding of the advantages and drawbacks of more realistic problem modelling can help guide their decision-making. Schedules that are created using a more accurate model can more easily be used, as many of the complicated constraints are taken into account. Besides, realistic modelling of passenger hinder allows better to reduce this hinder, which can have large societal benefits. However, this work also shows that increased accuracy requires large and complex objective calculations, and limits the available solving methods. To make sure good schedules are found, re-evaluating the algorithms is necessary when the problem definition undergoes changes.

Furthermore, some different real-world problems that need to be modelled as realistically as the Dutch railway maintenance scheduling problem can benefit from the conclusions from this thesis. Many of the common drawbacks of solving a real-world problem came forward in this process. Therefore, the general conclusions on improving the problem definition and the solving method could also be useful for different problems on this precise, complex level. There is limited work on problems of a similar scale and complexity. And since using domain knowledge is essential in such a complex function, no existing solving methods can be applied out of the box.

9.2 Future work

In this section, the most important future work directions from the content chapters are highlighted. Then, some additional general future work directions will be presented.

The main future research direction obtained from analysis and first improvements to the current algorithm is to further research the implementation of the current objective calculation. More inefficiencies in implementation and memory managements could be identified. For example, multiple aspects of the objective could benefit from being calculated only just before the schedule is evaluated. This means they are done based on multiple changes to the schedule, rather than re-evaluating after every change. Besides, some points where the model could be simplified were discovered. These have only limited effects on the expected solutions, but do impact the runtime. Most importantly, the computation and caching of costs per passenger stream could be reduced by simplifying the passenger hinder definition.

From the approximation of the detour paths, the main takeaway for future research would be to research a new definition of passenger routes, and specifically the transfer times. The current assumption that passengers always take the shortest path on this graph does not always correspond with the routes expected in reality. Furthermore, approximating the costs for travellers being delayed and their required alternative travel per hour, or even per project request, would be a potential way to reduce runtime further. This is expected to have a small negative impact on the solution quality. However, scheduling with this error might result in a final solution that is good enough to be helpful in some phases of scheduling.

From the comparison of different search strategies, a good deal of possible future research was identified. Each of the compared search strategies could be further researched. Most importantly, the hybrid greedy-evolutionary strategy could be improved by doing more hyperparameter analysis and creating better mutations. An iterated greedy algorithm could also be an interesting future research. Besides, some possible improvements to the greedy algorithm with look-aheads were identified from the analysis of the look-ahead behaviour. Integrating these look-aheads into the hybrid algorithm could also be investigated.

From the research into using an improved order for better solution quality with the greedy algorithm, the most important future research direction is to try other techniques for finding a new order. This could for example be an adaptive prioritisation, meaning that the next request to add to the schedule is selected based on the current partial solution. The proposed order function improved results, but has some limitations. It does not take into account all aspects of a problem's potential difficulty and is quite a simplified representation of a request's 'importance'. Besides, the simulated annealing is unable to reliably find good weights that improve solution quality. Therefore, further research to improve the design and optimisation of the order function is also interesting.

Finally, from the generalisation of the results to different input years, the main recommendation would be to research methods to create a more representative set of problems. These problems can be used to test different algorithms with. Currently, only limited historical data is available, and no synthetic representative problems have been created. Besides that, a few potential future research directions were identified from the results on different years. The most important one is to research different order methods and how they generalise to different input problem characteristics and solving methods.

Besides the future work identified in the content chapters, there were a few potential future research directions identified that are not directly related to any of the research questions:

- **Creating better human-machine interaction between schedulers and algorithm.** Currently, the interaction between decision-makers and the algorithm is very limited. The algorithm suggests a schedule based on the current configuration settings, or it evaluates a full schedule created by the scheduling team. However, there are many ways in which this cooperation between the schedulers and the schedule algorithms could be improved. For example, providing direct feedback of the change in cost from a scheduler-made mutation to the schedule. Or alternatively, asking a user to provide input on some decisions made by the greedy constructive algorithm, as a tie-breaker for example.
- **Improving robustness to possible unexpected maintenance.** Besides the preventive maintenance that can be determined for the full year ahead of time, there are also inevitably failures, which require corrective maintenance. Research into a suitable way to take robustness of the schedule to this corrective maintenance into account in the objective function could therefore increase the quality of how the schedule can finally be executed.
- **Pre-computing lowest hinder spots for certain lengths.** In general, the passenger travel is modelled to be very regular. The passenger travel at a certain day of the week and time is the same for all weeks outside the holidays. Therefore, it is possible to pre-compute good locations for scheduling a project request. Determining these good locations on a shorter scale, and then re-using these good spots on a larger scale could, for instance, make the solving faster.
- **Combining different levels of maintenance.** Currently, only the larger requests are scheduled within a yearly schedule. Shorter requests are often done during nights, and are scheduled on a shorter term, or using reservations [11], [13]. Incorporating this information in the scheduling could improve the way these two are aligned. Similarly, better combination of this level with the multi-year level of the maintenance schedule could improve the research schedules.
- **Coupling with timetabling.** An important trend identified in the field of maintenance scheduling is to couple it with the timetabling problem [20]. This is not directly applicable for this case, as the timetable is handled by different companies and is not done on the same scale. However, by integrating some timetable information, it is possible to determine situations where the passenger hinder could be further reduced by adapting the timetable or train routes to the maintenance work.
- **Increasing number of replications and hyperparameter tuning.** Due to the large computational resources required to create the yearly maintenance schedules, only a limited number of replications were done for all experiments. This does not allow identifying smaller differences between certain algorithms. Besides, more replications could increase confidence in the results. No complete hyperparameter analyses were done for all algorithms, so this could still improve solution quality further.

A Input properties of created sub-problems

For different experiments in this thesis, smaller size problem inputs were required. Therefore, synthetic problem instances were created by using a subset of the maintenance from an annual problem. This smaller set of maintenance then needs to be scheduled on a smaller period. The process of how these were created is explained in section 3.3.3. Some more detailed properties of every sub-problem are provided here.

A.1 2024

For the year of 2024, a single three-month scheduling problem was created to schedule from October 1st till December 31st: 2024-Q4. Besides that, six two-month scheduling problems were created: 2024-P1, 2024-P2, etc. The properties of these sub-problems are given in fig. A.1 and in table A.1.

Most properties are quite similar in all periods. It can be seen from this that periods later in the year have shorter time periods. This is due to the fact that almost all required time periods have more restricting start times. Therefore, the shorter time windows are scheduled later in the year. The periods later in the year also have a higher number of possible conflicts. This is expected to make these sub-problems more difficult to solve. Besides this, the 2024-P4 sub-problem has a smaller number of requests, so this is expected to be faster to solve.

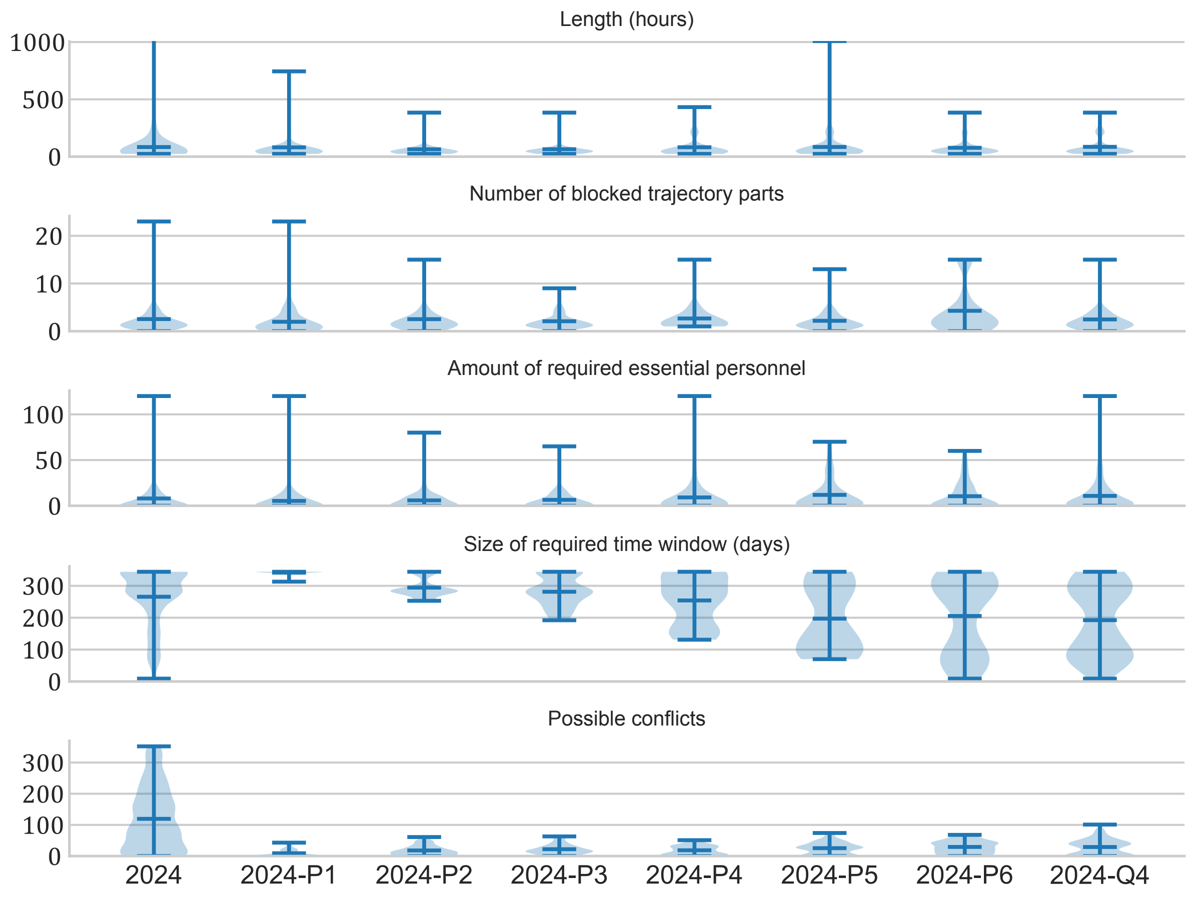


Figure A.1: Violin plot of the distribution of the most important problem characteristics for all (sub-)problems for the scheduling year 2024.

		2024	P1	P2	P3	P4	P5	P6	Q4
Projects	total	729	132	139	138	88	131	100	188
	in cluster	11	0	2	1	3	3	2	3
	with PEAT	414	84	74	70	45	82	59	118
	blocking	682	98	134	137	88	126	98	178
Length (hours)	mean	83.6	80.9	64.1	64.7	81.8	84.3	77.0	85.5
	min	24	24	24	24	24	24	24	24
	Q1	48	48	48	48	48	48	48	48
	median	48	48	48	48	48	48	48	48
	Q3	48	48	48	48	48	48	72	54
	max	6480	744	384	384	432	1008	384	384
Blocked trajectory parts	mean	2.5	2.0	2.5	2.1	2.7	2.2	4.3	2.5
	min	0	0	0	0	1	0	0	0
	Q1	1	0	1	1	1	1	1	1
	median	2	1	2	2	2	1	2	2
	Q3	3	2	3	2	3	3	4	3
	max	23	23	15	9	15	13	15	15
Essential personnel	mean	8.0	5.3	5.9	6.5	9.1	11.9	10.4	10.8
	min	0	0	0	0	0	0	0	0
	Q1	0	0	0	0	0	0	0	0
	median	0	0	5	5	0	5	0	5
	Q3	10	10	10	10	10	15	20	15
	max	120	120	80	65	120	70	60	120
Time window size (days)	mean	265.7	341.2	294.4	281.3	254.1	197.1	205.4	192.3
	min	9	313	253	192	131	70	9	9
	Q1	223	344	284	253	162	100	70	70
	median	284	344	284	284	284	162	284	177
	Q3	344	344	313	344	344	284	313	284
	max	344	344	344	344	344	344	344	344
Possible conflicts	mean	119.4	9.0	18.0	22.0	18.5	25.3	29.2	29.0
	min	0	0	0	0	0	0	0	0
	Q1	25	0	2.5	11	6	4	11	4
	median	97	0	14	18	20	27	25	31.5
	Q3	193.5	17	28	31	31.2	39	44	43.2
	max	352	43	61	63	51	74	68	101
Trajectory parts	total	344	344	344	344	344	344	344	344
	blocked	200	89	115	105	98	95	109	132
Requests blocking trajectory part	mean	7.5	2.1	2.9	2.7	2.4	2.9	3.9	3.3
	min	0	0	0	0	1	0	0	0
	Q1	1	0	1	1	1	1	1	1
	median	5	1	3	2	2	2	2	3
	Q3	11	3	4	3	3	4	4	5
	max	40	14	8	13	8	9	15	15

Table A.1: The most important characteristics of the maintenance work in sub-problems of the year 2024. A request with personnel means that a non-zero amount of personnel costs is specified for that project request. A blocking project request means that it blocks at least one trajectory part. The blocked trajectory parts are the number of blocked trajectory part by a certain request. The essential personnel is the sum of the three types of essential personnel. The time window size is the rounded number of days that is available within the required start time and required end time of the request. The possible conflicts are the sum of dependencies, pre-planned hinder and other request with which a project request can not be planned overlapping. The request blocking trajectory part is the sum of requests which block a certain trajectory part, these values are therefore a distribution over all trajectory parts.

A.2 Additional scheduling years

Most experiments were done with in schedule input for the year 2024. In chapter 8, the most promising techniques were also applied on the (sub-)problems for the years 2023 and 2025. These sub-problems were created in the same manner as for 2024. Some more detailed properties of these sub-problems are presented here.

A.2.1 2023

For 2023, there are four sub-periods for three-month schedules. The properties of these sub-problems as well as the full year problem are presented in fig. A.2 and in table A.2.

The third quarter has the longest requests and the most blocked trajectory parts. There are also the highest mean number of requests blocking the same trajectory pars. The fourth quarter blocks the most different trajectory parts. In three of the periods, there are some very small time windows. The essential personnel demands are more equally spread over the different quarters. Most time windows are very large for the first quarter, the other three quarters have more restrictive time windows. The number of possible conflicts becomes higher towards the end of the year.

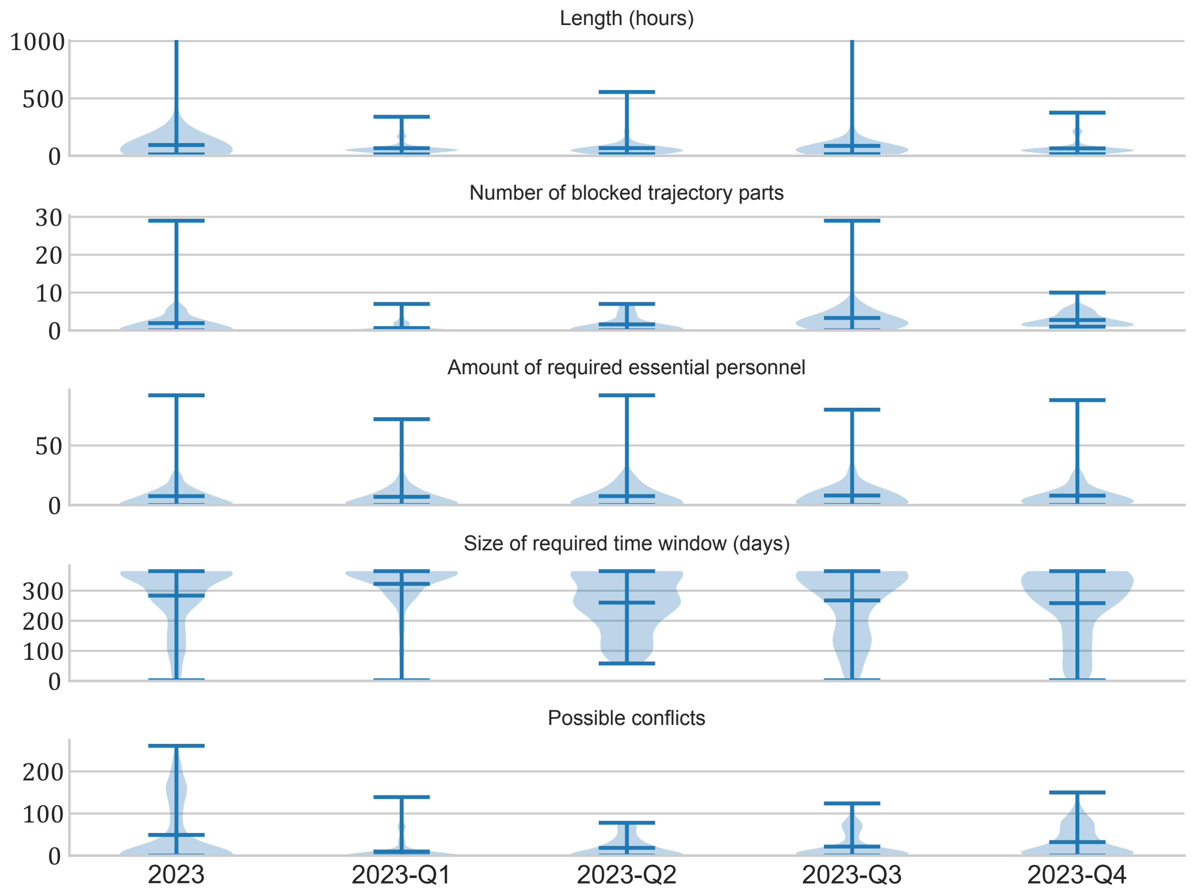


Figure A.2: Violin plot of the distribution of the most important problem characteristics for all (sub-)problems for the scheduling year 2023.

		2023	Q1	Q2	Q3	Q4
Project requests	total	539	189	88	115	145
	in cluster	0	0	0	0	0
	with PEAT	463	159	80	101	121
	blocking	353	49	53	105	145
Length (hours)	mean	93.6	66.2	66.8	86.3	64.6
	min	12	12	14	14	14
	Q1	50	52	32	52	48
	median	52	52	52	52	52
	Q3	52	53	52	52	52
	max	6385	340	556	1223	376
Blocked trajectory parts	mean	1.9	0.6	1.6	3.3	2.8
	min	0	0	0	0	1
	Q1	0	0	0	1	1
	median	1	0	1	2	2
	Q3	3	1	2	3	4
	max	29	7	7	29	10
Essential personnel	mean	7.5	7.0	7.5	8.0	7.9
	min	0	0	0	0	0
	Q1	0	0	0	0	0
	median	5	4	3	6	5
	Q3	10	10	10.2	12	10
	max	92	72	92	80	88
Time window size (days)	mean	283.6	322.7	260.2	267.7	258.5
	min	2	2	58	2	2
	Q1	245.5	304	198	155	184
	median	321	351	274	304	290
	Q3	365	365	365	365	348
	max	365	365	365	365	365
Possible conflicts	mean	49.1	9.6	18.3	21.4	32.1
	min	0	0	0	0	0
	Q1	0	0	0	4	4
	median	4	0	4	4	15
	Q3	91	0	27	22	55
	max	261	139	78	124	150
Trajectory parts	total	344	344	344	344	344
	blocked	239	73	62	97	123
Requests blocking trajectory part	mean	2.4	0.5	1.5	3.5	3.2
	min	0	0	0	0	1
	Q1	0	0	0	1	1
	median	1	0	1	2	2
	Q3	3	1	2	3	3.5
	max	64	8	8	26	27

Table A.2: The most important characteristics of the maintenance work in sub-problems of the year 2023. A request with personnel means that a non-zero amount of personnel costs is specified for that project request. A blocking project request means that it blocks at least one trajectory part. The blocked trajectory parts are the number of blocked trajectory part by a certain request. The essential personnel is the sum of the three types of essential personnel. The time window size is the rounded number of days that is available within the required start time and required end time of the request. The possible conflicts are the sum of dependencies, pre-planned hinder and other request with which a project request can not be planned overlapping. The request blocking trajectory part is the sum of requests which block a certain trajectory part, these values are therefore a distribution over all trajectory parts.

A.2.2 2025

There are four sub-periods created from the 2025 schedule year. The distribution of the general properties of these sub-problems are presented in table A.3. The most important ones are also visually presented as a violin plot in fig. A.3.

In the properties from the 2025 sub-problems, it can be seen that the length, blocked trajectory parts and possible conflicts are distributed more equally than in the other years. The size of the required time window is smallest in later quarters, similarly to the year 2024. This is explained by the restrictive start times. The requests are also quite unevenly spread over the different quarters. The fourth quarter contains by far the most requests, and these requests also have the most possible conflicts.

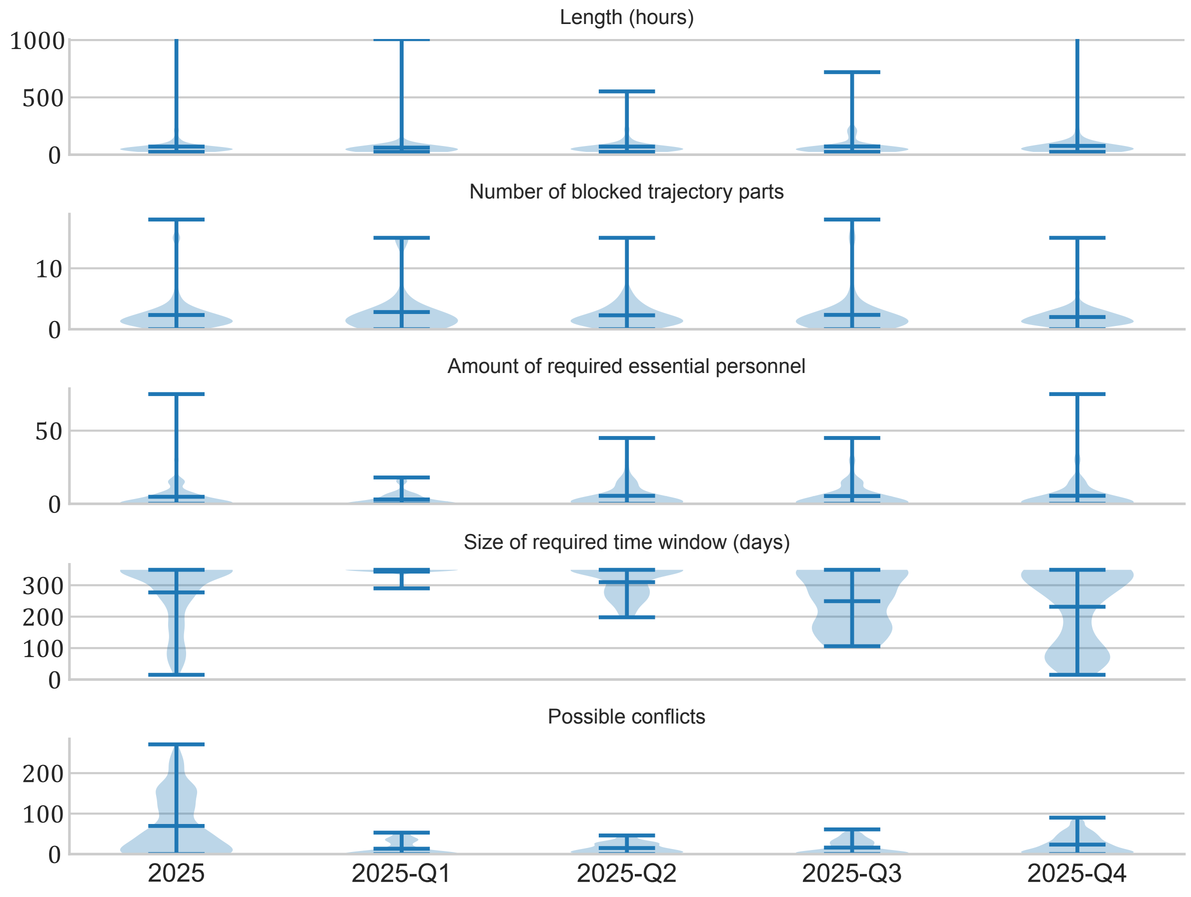


Figure A.3: Violin plot of the distribution of the most important problem characteristics for all (sub-)problems for the scheduling year 2025.

		2025	Q1	Q2	Q3	Q4
Project requests	total	597	151	104	126	216
	in cluster	117	18	24	31	46
	with PEAT	226	57	36	49	84
	blocking	540	129	97	112	203
Length (hours)	mean	70.7	61.7	70.6	71.6	76.7
	min	24	24	24	24	24
	Q1	48	48	48	48	48
	median	48	48	48	48	48
	Q3	48	48	48	48	72
	max	1224	1008	552	720	1224
Blocked trajectory parts	mean	2.3	2.8	2.3	2.4	2.0
	min	0	0	0	0	0
	Q1	1	1	1	1	1
	median	2	2	2	1	2
	Q3	3	3	3	3	3
	max	18	15	15	18	15
Essential personnel	mean	4.8	3.0	5.5	5.3	5.5
	min	0	0	0	0	0
	Q1	0	0	0	0	0
	median	3	0	3	3	3
	Q3	6	4.5	6	6	6
	max	75	18	45	45	75
Time window size (days)	mean	277.1	343.3	309.8	249.2	231.5
	min	15	290	198	106	15
	Q1	229	349	259	168	106
	median	349	349	349	259	290
	Q3	349	349	349	349	349
	max	349	349	349	349	349
Possible conflicts	mean	69.5	13.3	15.0	16.0	23.5
	min	0	0	0	0	0
	Q1	2	0	5	2.5	6
	median	46	4	12.5	7	16
	Q3	118	25.5	26	28	38
	max	271	53	46	61	90
Trajectory parts	total	344	344	344	344	344
	blocked	197	98	99	100	136
Requests blocking trajectory part	mean	5.5	3.5	2.2	2.6	2.9
	min	0	0	0	0	0
	Q1	1	1	1	1	1
	median	3	2	2	2	2
	Q3	8	5	3	4	4
	max	22	13	8	8	11

Table A.3: The most important characteristics of the maintenance work in sub-problems of the year 2025. A request with personnel means that a non-zero amount of personnel costs is specified for that project request. A blocking project request means that it blocks at least one trajectory part. The blocked trajectory parts are the number of blocked trajectory part by a certain request. The essential personnel is the sum of the three types of essential personnel. The time window size is the rounded number of days that is available within the required start time and required end time of the request. The possible conflicts are the sum of dependencies, pre-planned hinder and other request with which a project request can not be planned overlapping. The request blocking trajectory part is the sum of requests which block a certain trajectory part, these values are therefore a distribution over all trajectory parts.

B

Convergence visualisation of different search strategies

In this chapter, some extra figures visualising the behaviour of different search strategies are given.

B.1 Evolutionary algorithm

Two repetitions of the evolutionary algorithm are visualised in fig. B.1.

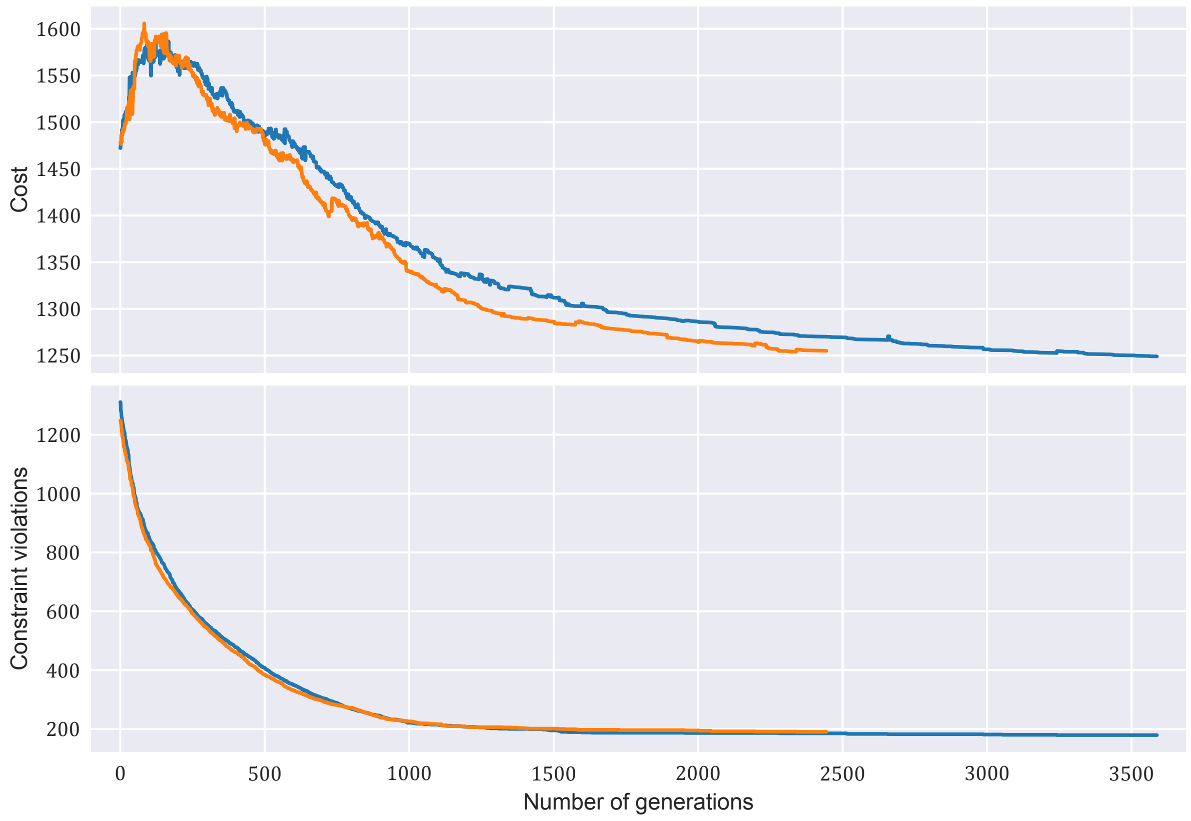


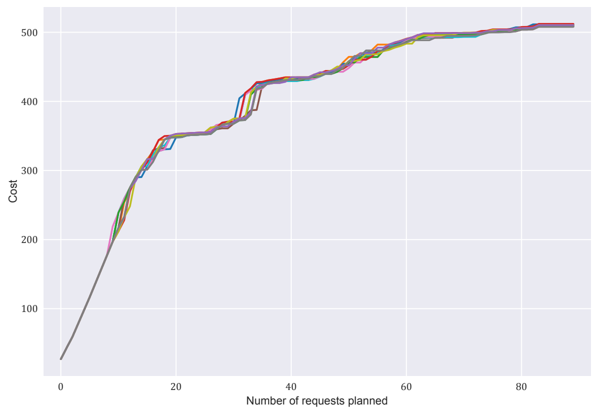
Figure B.1: Two repetitions of the evolutionary algorithm. The orange line was done for 32 hours, the blue run was 48 hours. Both results are substantially worse than each of the other algorithms.

B.2 Hybrid greedy-evolutionary

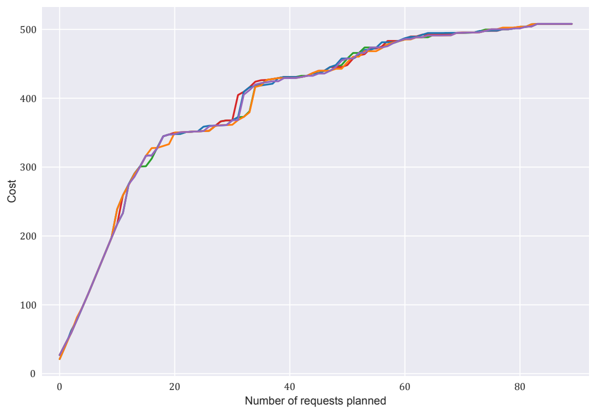
The hybrid greed-evolutionary algorithm was replicated five times on the year of 2024, resulting in the best schedules. The results of every step of the algorithm is separately visualised in this section, for both phases.

B.2.1 Phase 1

In the first phase, ten greedy runs are done with the first 100 requests. These are used as a starting population for 5000 generations of evolutionary search. In fig. B.2a, the ten different runs for one of the replications are given. In fig. B.2b, the best out of these ten was visualised for each of the five replications.



(a) Ten different greedy heuristic runs to create a starting population for evolutionary search.



(b) The best of the ten greedy starting individuals for five replications of the hybrid greedy-evolutionary algorithm.

Figure B.2: Creating an evolutionary starting population using the greedy algorithm.

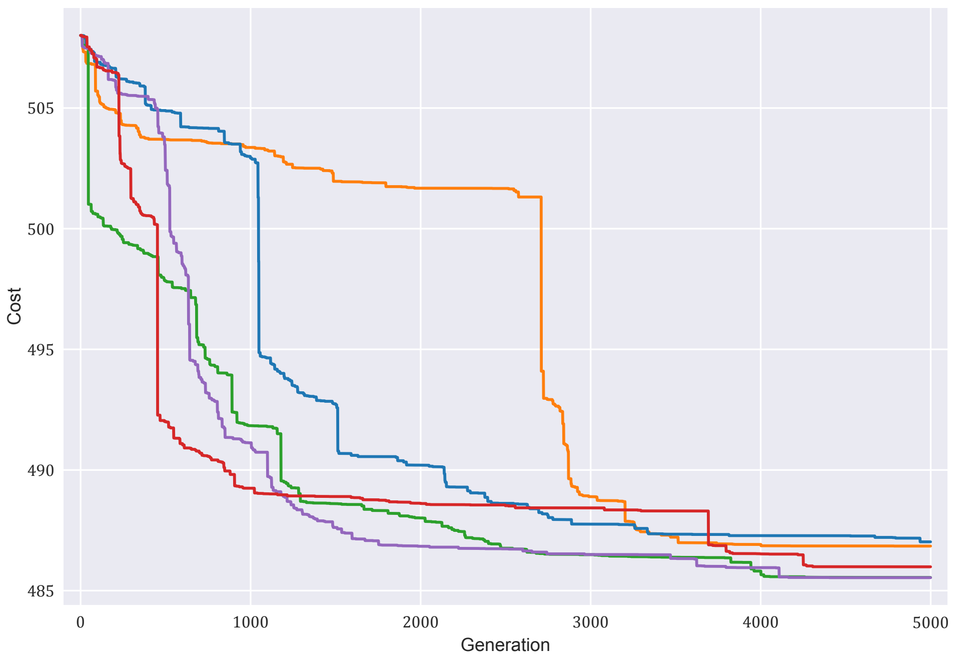


Figure B.3: The first phase of evolutionary search in the hybrid greedy-evolutionary algorithm.

B.2.2 Phase 2

In the second phase, the best individual at the end of the first evolutionary search is used as a starting point. First, the rest of the project requests are scheduled greedily. The process of this is visualised in fig. B.4. Finally, the result of this greedy process is cloned to create ten individuals, as a starting population for the second round of evolutionary search. The convergence of this part is visualised in fig. B.5.

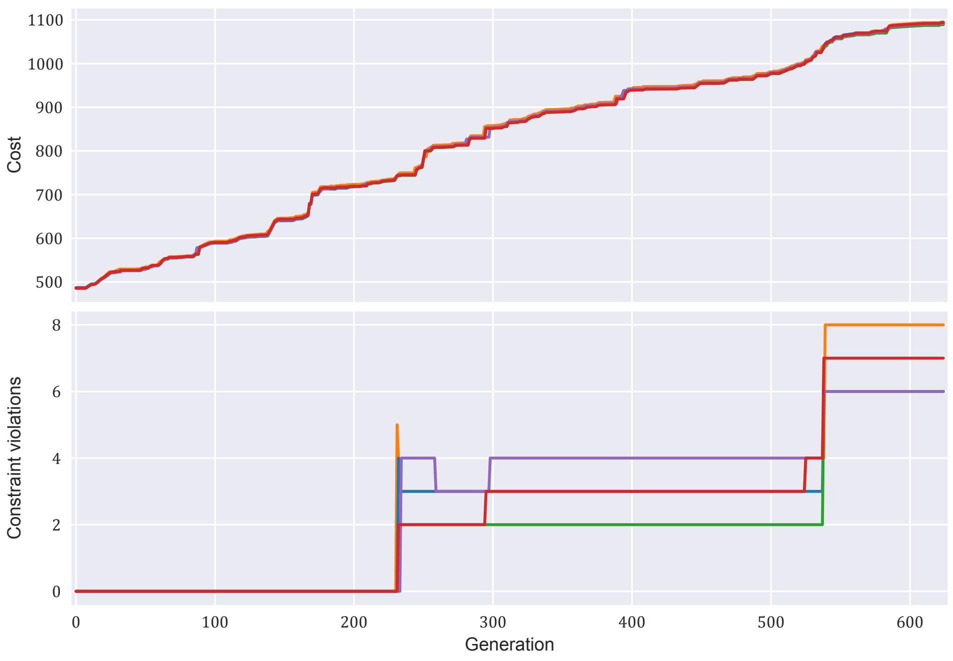


Figure B.4: The error of different detour path approximations per hour on an isolated test set. The mean difference in the sum of the passenger and alternative travel costs is used.

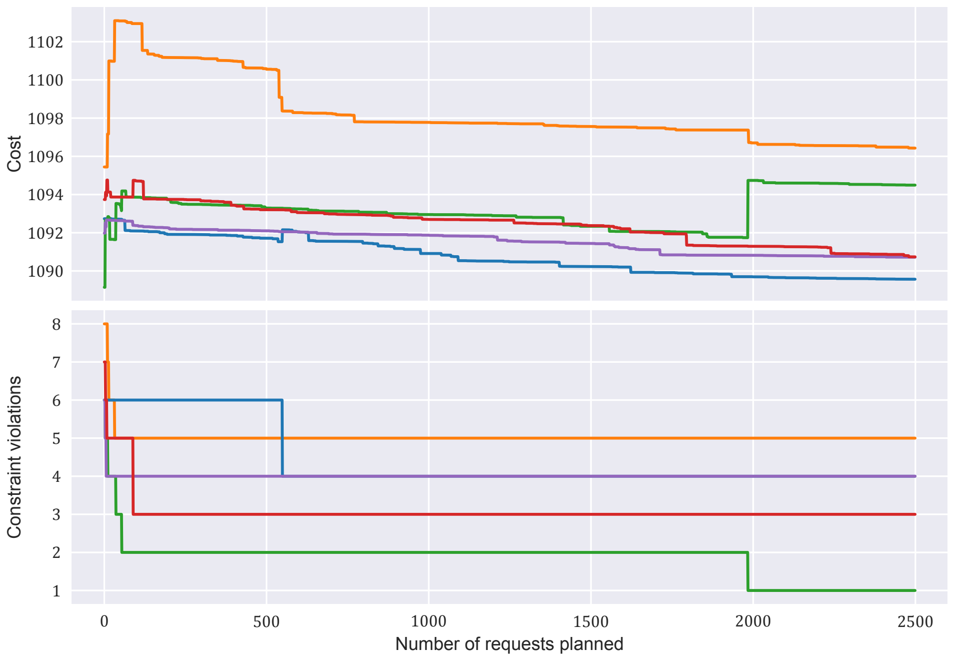


Figure B.5: The error of different detour path approximations per hour on an isolated test set. The mean difference in the sum of the passenger and alternative travel costs is used.

Bibliography

[1] ProRail, *ProRail Jaarverslag 2022*, nl, 2022. [Online]. Available: https://www.jaarverslagprorail.nl/FbContent.ashx/pub_1001/downloads/v230502111910/ProRail_Jaarverslag_2022.pdf (visited on 09/07/2023).

[2] "Ontwikkelagenda Toekomstbeeld OV," nl, Ministerie van Infrastuctuur en Waterstaat, Tech. Rep., Jan. 2021. [Online]. Available: https://open.overheid.nl/documenten/ronl-2311ee8d-89c9-4278-9f75-8dd8f3e4db51/pdf (visited on 09/01/2023).

[3] Dennistw, *Nederlands: Railway map the netherlands*, Sep. 2017. [Online]. Available: https://commons.wikimedia.org/w/index.php?curid=62275026 (visited on 10/06/2023).

[4] "Eleventh Annual Market Monitoring Report," Independent Regulator's Group - Rail, Tech. Rep. 11, Apr. 2023. [Online]. Available: https://irg-rail.eu/download/5/957/IRG-Rail-11thMMReport-Mainreport.pdf.

[5] M. Oudshoorn, "Solving a real-world rail maintenance scheduling problem," en, M.S. thesis, Delft University of Technology, Jul. 2019. [Online]. Available: http://resolver.tudelft.nl/uuid:95e95789-19d9-4c86-b386-00f147306bbe.

[6] M. Oudshoorn, T. Koppenberg, and N. Yorke-Smith, "Optimization of annual planned rail maintenance," en, *Computer-Aided Civil and Infrastructure Engineering*, vol. 37, no. 6, pp. 669–687, 2022, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/mice.12764, issn: 1467-8667. doi: 10.1111/mice.12764. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/mice.12764 (visited on 02/02/2023).

[7] A. R. Jenema, "An optimization model for a Train-Free-Period planning for ProRail based on the maintenance needs of the Dutch railway infrastructure," en, M.S. thesis, Sep. 2011.

[8] Y. de Weert, "Improving the scheduling of railway maintenance projects by considering passenger hindrance and event requests of passenger operators," en, M.S. thesis, Jun. 2022.

[9] D. den Hertog, J. I. van Zante-de Fokkert, S. A. Sjamaar, and R. Beusmans, "Optimal working zone division for safe track maintenance in The Netherlands," *Accident Analysis & Prevention*, vol. 37, no. 5, pp. 890–893, Sep. 2005, issn: 0001-4575. doi: 10.1016/j.aap.2005.04.006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0001457505000680 (visited on 09/10/2023).

[10] J. I. v. Zante–de Fokkert, D. den Hertog, F. J. v. d. Berg, and J. H. M. Verhoeven, "The Netherlands Schedules Track Maintenance to Improve Track Workers' Safety," *Interfaces*, vol. 37, no. 2, pp. 133–142, Apr. 2007, Publisher: INFORMS, issn: 0092-2102. doi: 10.1287/inte.1060.0246. [Online]. Available: https://pubsonline.informs.org/doi/abs/10.1287/inte.1060.0246 (visited on 09/10/2023).

[11] F. Nijland, K. Gkiotsalitis, and E. C. van Berkum, "Improving railway maintenance schedules by considering hindrance and capacity constraints," en, *Transportation Research Part C: Emerging Technologies*, vol. 126, p. 103 108, May 2021, issn: 0968-090X. doi: 10.1016/j.trc.2021.103108. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0968090X21001273 (visited on 02/07/2023).

[12] B. Buurman, "Railway maintenance reservation scheduling considering train traffic and maintenance demand," M.S. thesis, University of Twente, 2021.

[13] B. Buurman, K. Gkiotsalitis, and E. C. van Berkum, "Railway maintenance reservation scheduling considering detouring delays and maintenance demand," en, *Journal of Rail Transport Planning & Management*, vol. 25, p. 100 359, Mar. 2023, issn: 2210-9706. doi: 10.1016/j.jrtpm.2022.100359. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2210970622000592 (visited on 02/03/2023).

[14] G. Budai-Balke, *Operations research models for scheduling railway infrastructure maintenance*. Rozenberg Publishers, 2009.

[15] H. Pouryousef, P. Teixeira, and J. Sussman, “Track maintenance scheduling and its interactions with operations: Dedicated and mixed high-speed rail (hsr) scenarios,” in *Joint Rail Conference*, vol. 49071, 2010, pp. 317–326.

[16] N. Bešinović, B. Widarno, and R. M. Goverde, “Adjusting freight train paths to infrastructure possessions,” in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, Sep. 2020, pp. 1–6. doi: 10.1109/ITSC45102.2020.9294192.

[17] J. Trepat Borecka and N. Bešinović, “Scheduling multimodal alternative services for managing infrastructure maintenance possessions in railway networks,” *Transportation Research Part B: Methodological*, vol. 154, pp. 147–174, Dec. 2021, issn: 0191-2615. doi: 10.1016/j.trb.2021.10.009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0191261521001946 (visited on 09/11/2023).

[18] T. Lidén, “Railway Infrastructure Maintenance - A Survey of Planning Problems and Conducted Research,” en, *Transportation Research Procedia*, 18th Euro Working Group on Transportation, EWGT 2015, 14-16 July 2015, Delft, The Netherlands, vol. 10, pp. 574–583, Jan. 2015, issn: 2352-1465. doi: 10.1016/j.trpro.2015.09.011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352146515001982 (visited on 02/07/2023).

[19] T. Lidén, “Survey of railway maintenance activities from a planning perspective and literature review concerning the use of mathematical algorithms for solving such planning and scheduling problems,” en, 2014.

[20] M. Sedghi, O. Kauppila, B. Bergquist, E. Vanhatalo, and M. Kulahci, “A taxonomy of railway track maintenance planning and scheduling: A review and research trends,” en, *Reliability Engineering & System Safety*, vol. 215, p. 107 827, Nov. 2021, issn: 0951-8320. doi: 10.1016/j.ress.2021.107827. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0951832021003483 (visited on 02/06/2023).

[21] B. S. N. Cheung, K. P. Chow, L. C. K. Hui, and A. M. K. Yong, “Railway track possession assignment using constraint satisfaction,” *Engineering Applications of Artificial Intelligence*, vol. 12, no. 5, pp. 599–611, Oct. 1999, issn: 0952-1976. doi: 10.1016/S0952-1976(99)00025-1. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0952197699000251 (visited on 09/10/2023).

[22] F. Peng, S. Kang, X. Li, Y. Ouyang, K. Somani, and D. Acharya, “A Heuristic Approach to the Railroad Track Maintenance Scheduling Problem,” en, *Computer-Aided Civil and Infrastructure Engineering*, vol. 26, no. 2, pp. 129–145, 2011, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8667.2010.00670.x, issn: 1467-8667. doi: 10.1111/j.1467-8667.2010.00670.x. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8667.2010.00670.x (visited on 02/07/2023).

[23] F. Peng and Y. Ouyang, “Track maintenance production team scheduling in railroad networks,” *Transportation Research Part B: Methodological*, vol. 46, no. 10, pp. 1474–1488, Dec. 2012, issn: 0191-2615. doi: 10.1016/j.trb.2012.07.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0191261512000975 (visited on 09/13/2023).

[24] N. Boland, T. Kalinowski, H. Waterer, and L. Zheng, “Mixed integer programming based maintenance scheduling for the Hunter Valley coal chain,” en, *Journal of Scheduling*, vol. 16, no. 6, pp. 649–659, Dec. 2013, issn: 1094-6136, 1099-1425. doi: 10.1007/s10951-012-0284-y. [Online]. Available: http://link.springer.com/10.1007/s10951-012-0284-y (visited on 02/07/2023).

[25] N. Boland, T. Kalinowski, H. Waterer, and L. Zheng, “Scheduling arc maintenance jobs in a network to maximize total flow over time,” *Discrete Applied Mathematics*, Matheuristics 2010, vol. 163, pp. 34–52, Jan. 2014, issn: 0166-218X. doi: 10.1016/j.dam.2012.05.027. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0166218X12002338 (visited on 09/13/2023).

[26] M. Forsgren, M. Aronsson, and S. Gestrelius, “Maintaining tracks and traffic flow at the same time,” en, *Journal of Rail Transport Planning & Management*, Robust Rescheduling and Capacity Use, vol. 3, no. 3, pp. 111–123, Aug. 2013, issn: 2210-9706. doi: 10.1016/j.jrtpm.2013.11.001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2210970613000267 (visited on 02/07/2023).

[27] S. Khalouli, R. Benmansour, and S. Hanafi, “An ant colony algorithm based on opportunities for scheduling the preventive railway maintenance,” in *2016 International Conference on Control, Decision and Information Technologies (CoDIT)*, Apr. 2016, pp. 594–599. doi: 10.1109/CoDIT.2016.7593629.

[28] X. Luan, J. Miao, L. Meng, F. Corman, and G. Lodewijks, “Integrated optimization on train scheduling and preventive maintenance time slots planning,” en, *Transportation Research Part C: Emerging Technologies*, vol. 80, pp. 329–359, Jul. 2017, issn: 0968-090X. doi: 10.1016/j.trc.2017.04.010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0968090X17301237 (visited on 02/07/2023).

[29] T. Lidén and M. Joborn, “An optimization model for integrated planning of railway traffic and network maintenance,” en, *Transportation Research Part C: Emerging Technologies*, vol. 74, pp. 327–347, Jan. 2017, issn: 0968-090X. doi: 10.1016/j.trc.2016.11.016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0968090X16302340 (visited on 02/07/2023).

[30] T. Lidén, *Concurrent planning of railway maintenance windows and train services*, en. Linköping University Electronic Press, Nov. 2018, Google-Books-ID: Fsp6DwAAQBAJ, isbn: 978-91-7685-201-9.

[31] T. Lidén, “Coordinating maintenance windows and train traffic: A case study,” en, *Public Transport*, vol. 12, no. 2, pp. 261–298, Jun. 2020, issn: 1613-7159. doi: 10.1007/s12469-020-00232-2. [Online]. Available: https://doi.org/10.1007/s12469-020-00232-2 (visited on 02/07/2023).

[32] C. Zhang, Y. Gao, L. Yang, U. Kumar, and Z. Gao, “Integrated optimization of train scheduling and maintenance planning on high-speed railway corridors,” en, *Omega*, vol. 87, pp. 86–104, Sep. 2019, issn: 0305-0483. doi: 10.1016/j.omega.2018.08.005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0305048317310575 (visited on 02/07/2023).

[33] A. D’Ariano, L. Meng, G. Centulio, and F. Corman, “Integrated stochastic optimization approaches for tactical scheduling of trains and railway infrastructure maintenance,” en, *Computers & Industrial Engineering*, vol. 127, pp. 1315–1335, Jan. 2019, issn: 0360-8352. doi: 10.1016/j.cie.2017.12.010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360835217305764 (visited on 02/03/2023).

[34] C. Zhang, Y. Gao, L. Yang, Z. Gao, and J. Qi, “Joint optimization of train scheduling and maintenance planning in a railway network: A heuristic algorithm using Lagrangian relaxation,” en, *Transportation Research Part B: Methodological*, vol. 134, pp. 64–92, Apr. 2020, issn: 0191-2615. doi: 10.1016/j.trb.2020.02.008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0191261519300840 (visited on 02/07/2023).

[35] Y. Chang, R. Liu, and Y. Tang, “Segment-condition-based railway track maintenance schedule optimization,” en, *Computer-Aided Civil and Infrastructure Engineering*, vol. 38, no. 2, pp. 160–193, 2023, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/mice.12824, issn: 1467-8667. doi: 10.1111/mice.12824. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/mice.12824 (visited on 02/02/2023).

[36] M. Sedghi, B. Bergquist, E. Vanhatalo, and A. Migdalas, “Data-driven maintenance planning and scheduling based on predicted railway track condition,” en, *Quality and Reliability Engineering International*, vol. 38, no. 7, pp. 3689–3709, 2022, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/qre.3166, issn: 1099-1638. doi: 10.1002/qre.3166. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/qre.3166 (visited on 02/03/2023).

[37] R. Mohammadi and Q. He, “A deep reinforcement learning approach for rail renewal and maintenance planning,” en, *Reliability Engineering & System Safety*, vol. 225, p. 108 615, Sep. 2022, issn: 0951-8320. doi: 10.1016/j.ress.2022.108615. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0951832022002575 (visited on 02/03/2023).

[38] A. Kasraei and J. Ali Zakeri, “Maintenance Decision Support Model for Railway Track Geometry Maintenance Planning Using Cost, Reliability, and Availability Factors: A Case Study,” en, *Transportation Research Record: Journal of the Transportation Research Board*, vol. 2676, no. 7, pp. 161–172, Jul. 2022, issn: 0361-1981, 2169-4052. doi: 10.1177/03611981221077089. [Online]. Available: http://journals.sagepub.com/doi/10.1177/03611981221077089 (visited on 02/03/2023).

[39] I. Stipanovic, Z. A. Bukhsh, C. Reale, and K. Gavin, “A multiobjective decision□making model for risk□based maintenance scheduling of railway earthworks,” en, *Applied Research on English Language*, vol. 11, no. 3, 2021, issn: 2252-0198. doi: 10.3390/app11030965. [Online]. Available: https://repository.tudelft.nl/islandora/object/uuid%3A621bd481-1d62-428b-8a03-a4fa59dacc7f (visited on 02/03/2023).

[40] R. Mohammadi, Q. He, and M. Karwan, “Data-driven robust strategies for joint optimization of rail renewal and maintenance planning,” en, *Omega*, vol. 103, p. 102 379, Sep. 2021, issn: 0305-0483. doi: 10.1016/j.omega.2020.102379. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0305048320307337 (visited on 02/06/2023).

[41] A. Consilvio, A. D. Febbraro, and N. Sacco, “A Rolling-Horizon Approach for Predictive Maintenance Planning to Reduce the Risk of Rail Service Disruptions,” *IEEE Transactions on Reliability*, vol. 70, no. 3, pp. 875–886, Sep. 2021, Conference Name: IEEE Transactions on Reliability, issn: 1558-1721. doi: 10.1109/TR.2020.3007504.

[42] T. Kalinowski, J. Matthews, and H. Waterer, “Scheduling of maintenance windows in a mining supply chain rail network,” en, *Computers & Operations Research*, vol. 115, p. 104 670, Mar. 2020, issn: 0305-0548. doi: 10.1016/j.cor.2019.03.016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0305054819300760 (visited on 02/07/2023).

[43] Q. Zhang, R. M. Lusby, P. Shang, and X. Zhu, “A heuristic approach to integrate train timetabling, platforming, and railway network maintenance scheduling decisions,” en, *Transportation Research Part B: Methodological*, vol. 158, pp. 210–238, Apr. 2022, issn: 0191-2615. doi: 10.1016/j.trb.2022.02.002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0191261522000212 (visited on 02/03/2023).

[44] M. Bababeik, M. Farjadamin, N. Khademi, and A.-H. Fani, “Simultaneous schedule of trains and track maintenance according to stochastic blockage time,” en, *International Journal of Rail Transportation*, vol. 10, no. 5, pp. 562–580, Sep. 2022, issn: 2324-8378, 2324-8386. doi: 10.1080/23248378.2021.1978884. [Online]. Available: https://www.tandfonline.com/doi/full/10.1080/23248378.2021.1978884 (visited on 02/03/2023).

[45] Jon Kleinberg and Eva Tardos, “Greedy Algorithms,” eng, in *Algorithm Design*, 1st ed., Pearson, 2005, pp. 115–183, isbn: 978-0-321-29535-4. [Online]. Available: http://archive.org/details/AlgorithmDesign1stEditionByJonKleinbergAndEvaTardos2005PDF (visited on 09/26/2023).

[46] D. K. Pradhan, J. Chakraborty, P. Choudhary, and S. Nandi, “An automated conflict of interest based greedy approach for conference paper assignment system,” *Journal of Informetrics*, vol. 14, no. 2, p. 101 022, May 2020, issn: 1751-1577. doi: 10.1016/j.joi.2020.101022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1751157719301373 (visited on 09/14/2023).

[47] W. Li, W. Huang, D. Jiang, and X. Liu, “A heuristic algorithm for cube packing with time schedule,” en, *Science in China Series F: Information Sciences*, vol. 53, no. 1, pp. 18–29, Jan. 2010, issn: 1869-1919. doi: 10.1007/s11432-010-0022-z. [Online]. Available: https://doi.org/10.1007/s11432-010-0022-z (visited on 09/14/2023).

[48] M. Huysmans, K. Coolen, F. Talla Nobibon, and R. Leus, “A fast greedy heuristic for scheduling modular projects,” en, *Journal of Heuristics*, vol. 21, no. 1, pp. 47–72, Feb. 2015, issn: 1572-9397. doi: 10.1007/s10732-014-9272-z. [Online]. Available: https://doi.org/10.1007/s10732-014-9272-z (visited on 09/14/2023).

[49] A. G. Lagodimos and V. Leopoulos, “Greedy heuristic algorithms for manpower shift planning,” *International Journal of Production Economics*, vol. 68, no. 1, pp. 95–106, Oct. 2000, issn: 0925-5273. doi: 10.1016/S0925-5273(99)00099-7. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925527399000997 (visited on 09/14/2023).

[50] Ahmed Abdulmunem Hussein, Esam Taha Yaseen, and Ahmed Noori Rashid, “Learnheuristics in routing and scheduling problems: A review,” en, *Samarra Journal of Pure and Applied Science*, vol. 5, no. 1, pp. 60–90, Mar. 2023, issn: 2789-6838, 2663-7405. doi: 10.54153/sjpas.2023.v5i1.445. [Online]. Available: https://sjpas.com/index.php/sjpas/article/view/445 (visited on 05/15/2023).

[51] R. Ruiz and T. Stützle, “A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem,” *European Journal of Operational Research*, vol. 177, no. 3, pp. 2033–2049, Mar. 2007, issn: 0377-2217. doi: 10.1016/j.ejor.2005.12.009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221705008507 (visited on 09/12/2023).

[52] V. Fernandez-Viagas, R. Ruiz, and J. M. Framinan, “A new vision of approximate methods for the permutation flowshop to minimise makespan: State-of-the-art and computational evaluation,” en, *European Journal of Operational Research*, vol. 257, no. 3, pp. 707–721, Mar. 2017, issn: 03772217. doi: 10.1016/j.ejor.2016.09.055. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0377221716308074 (visited on 09/12/2023).

[53] R. Ruiz, Q.-K. Pan, and B. Naderi, “Iterated Greedy methods for the distributed permutation flowshop scheduling problem,” *Omega*, vol. 83, pp. 213–222, Mar. 2019, issn: 0305-0483. doi: 10.1016/j.omega.2018.03.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0305048317306990 (visited on 09/12/2023).

[54] E. W. Dijkstra, “A note on two problems in connexion with graphs:(numerische mathematik, 1 (1959), p 269-271),” 1959. [Online]. Available: https://doi.org/10.1007/BF01386390.

[55] *Fundamentals of garbage collection - .NET*, en-us, Feb. 2023. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/standard/garbage-collection/fundamentals (visited on 09/29/2023).

[56] C. He, Y. Zhang, D. Gong, and X. Ji, “A review of surrogate-assisted evolutionary algorithms for expensive optimization problems,” en, *Expert Systems with Applications*, vol. 217, p. 119 495, May 2023, issn: 0957-4174. doi: 10.1016/j.eswa.2022.119495. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417422025143 (visited on 02/07/2023).

[57] J.-h. Hao, M. Liu, J.-h. Lin, and C. Wu, “A hybrid differential evolution approach based on surrogate modelling for scheduling bottleneck stages,” en, *Computers & Operations Research*, vol. 66, pp. 215–224, Feb. 2016, issn: 0305-0548. doi: 10.1016/j.cor.2015.08.005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0305054815002002 (visited on 05/15/2023).

[58] H. Bast, E. Carlsson, A. Eigenwillig, *et al.*, “Fast Routing in Very Large Public Transportation Networks Using Transfer Patterns.,” *ESA (1)*, vol. 6346, pp. 290–301, 2010.

[59] M. Holzer, F. Schulz, and D. Wagner, “Engineering multilevel overlay graphs for shortest-path queries,” *ACM Journal of Experimental Algorithmics*, vol. 13, 5:2.5–5:2.26, Feb. 2009, issn: 1084-6654. doi: 10.1145/1412228.1412239. [Online]. Available: https://doi.org/10.1145/1412228.1412239 (visited on 02/03/2023).

[60] H. Bast, S. Funke, D. Matijevic, P. Sanders, and D. Schultes, “In transit to constant time shortest-path queries in road networks,” in *2007 Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments (ALENEX)*, SIAM, 2007, pp. 46–59.

[61] H. Lu and Y. Shi, “Complexity of public transport networks,” *Tsinghua Science and Technology*, vol. 12, no. 2, pp. 204–213, Apr. 2007, Conference Name: Tsinghua Science and Technology, issn: 1007-0214. doi: 10.1016/S1007-0214(07)70027-5.

[62] H. Bast, D. Delling, A. Goldberg, *et al.*, “Route planning in transportation networks,” *Algorithm engineering: Selected results and surveys*, pp. 19–80, 2016.

[63] H.-G. Beyer and H.-P. Schwefel, “Evolution strategies – A comprehensive introduction,” en, *Natural Computing*, vol. 1, no. 1, pp. 3–52, Mar. 2002, issn: 1572-9796. doi: 10.1023/A:1015059928466. [Online]. Available: https://doi.org/10.1023/A:1015059928466 (visited on 09/06/2023).

[64] M. Emmerich, O. M. Shir, and H. Wang, “Evolution Strategies,” en, in *Handbook of Heuristics*, R. Martí, P. Panos, and M. G. C. Resende, Eds., Cham: Springer International Publishing, 2018, pp. 1–31, isbn: 978-3-319-07153-4. doi: 10.1007/978-3-319-07153-4_13-1. [Online]. Available: https://doi.org/10.1007/978-3-319-07153-4_13-1 (visited on 09/14/2023).

[65] M. Chen, “A Greedy Algorithm with Forward-Looking Strategy,” en, in *Greedy Algorithms*, IntechOpen, Nov. 2008, isbn: 978-953-7619-27-5. doi: 10.5772/6351. [Online]. Available: https://www.intechopen.com/chapters/5850 (visited on 09/14/2023).

[66] W. Q. Huang, Y. Li, H. Akeb, and C. M. Li, “Greedy algorithms for packing unequal circles into a rectangular container,” en, *Journal of the Operational Research Society*, vol. 56, no. 5, pp. 539–548, May 2005, issn: 1476-9360. doi: 10.1057/palgrave.jors.2601836. [Online]. Available: https://doi.org/10.1057/palgrave.jors.2601836 (visited on 09/14/2023).

[67] A. Ebrahimzadeh, N. Promwongsa, S. N. Afrasiabi, *et al.*, “H-Horizon Sequential Look-ahead Greedy Algorithm for VNF-FG Embedding,” in *2021 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Nov. 2021, pp. 41–46. doi: 10.1109/NFV-SDN53031.2021.9665063.

[68] D. Frost and R. Dechter, “Look-ahead value ordering for constraint satisfaction problems,” en, in *IJCAI (1)*, 1995, pp. 572–578.

[69] D. S. Nau, M. Luštrek, A. Parker, I. Bratko, and M. Gams, “When is it better not to look ahead?” *Artificial Intelligence*, vol. 174, no. 16, pp. 1323–1338, Nov. 2010, issn: 0004-3702. doi: 10.1016/j.artint.2010.08.002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370210001402 (visited on 09/12/2023).

[70] J. Pätzold, A. Schiewe, P. Schiewe, and A. Schöbel, “Look-Ahead Approaches for Integrated Planning in Public Transportation,” en, 16 pages, 2017, Artwork Size: 16 pages Medium: application/pdf Publisher: Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik GmbH, Wadern/Saarbruecken, Germany. doi: 10.4230/OASICS.ATMOS.2017.17. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2017/7894/ (visited on 09/14/2023).

[71] T. Bäck, “Evolution strategies: An alternative evolutionary algorithm,” in *European Conference on Artificial Evolution*, Springer, 1995, pp. 1–20.

[72] Y. Ye and A. Borodin, “Priority algorithms for the subset-sum problem,” en, *Journal of Combinatorial Optimization*, vol. 16, no. 3, pp. 198–228, Oct. 2008, issn: 1573-2886. doi: 10.1007/s10878-007-9126-9. [Online]. Available: https://doi.org/10.1007/s10878-007-9126-9 (visited on 09/12/2023).

[73] A. Borodin, M. N. Nielsen, and C. Rackoff, “(Incremental) Priority Algorithms,” en, *Algorithmica*, vol. 37, no. 4, pp. 295–326, Dec. 2003, issn: 1432-0541. doi: 10.1007/s00453-003-1036-3. [Online]. Available: https://doi.org/10.1007/s00453-003-1036-3 (visited on 09/12/2023).

[74] F. J. Gil-Gala, C. Mencía, M. R. Sierra, and R. Varela, “Evolving priority rules for on-line scheduling of jobs on a single machine with variable capacity over time,” en, *Applied Soft Computing*, vol. 85, p. 105 782, Dec. 2019, issn: 1568-4946. doi: 10.1016/j.asoc.2019.105782. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494619305630 (visited on 07/11/2023).

[75] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi, “Optimization by simulated annealing,” *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[76] R. Rutenbar, “Simulated annealing algorithms: An overview,” *IEEE Circuits and Devices Magazine*, vol. 5, no. 1, pp. 19–26, Jan. 1989, Conference Name: IEEE Circuits and Devices Magazine, issn: 1558-1888. doi: 10.1109/101.17235.

[77] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais, “Boosting Systematic Search by Weighting Constraints.,” Jan. 2004, pp. 146–150.

[78] T. Balafoutis and K. Stergiou, “On conflict-driven variable ordering heuristics,” in *Proceedings of Thirteenth Annual ERCIM International Workshop on Constraint Solving and Constraint Logic Programming, CSCLP-08*, 2008.

[79] S. Ozdemir and D. Susarla, *Feature Engineering Made Easy: Identify unique features from your dataset in order to build powerful machine learning systems*. Packt Publishing Ltd, 2018.

[80] M. □urasević and D. Jakobović, “Creating dispatching rules by simple ensemble combination,” en, *Journal of Heuristics*, vol. 25, no. 6, pp. 959–1013, Dec. 2019, issn: 1572-9397. doi: 10.1007/s10732-019-09416-x. [Online]. Available: https://doi.org/10.1007/s10732-019-09416-x (visited on 07/11/2023).