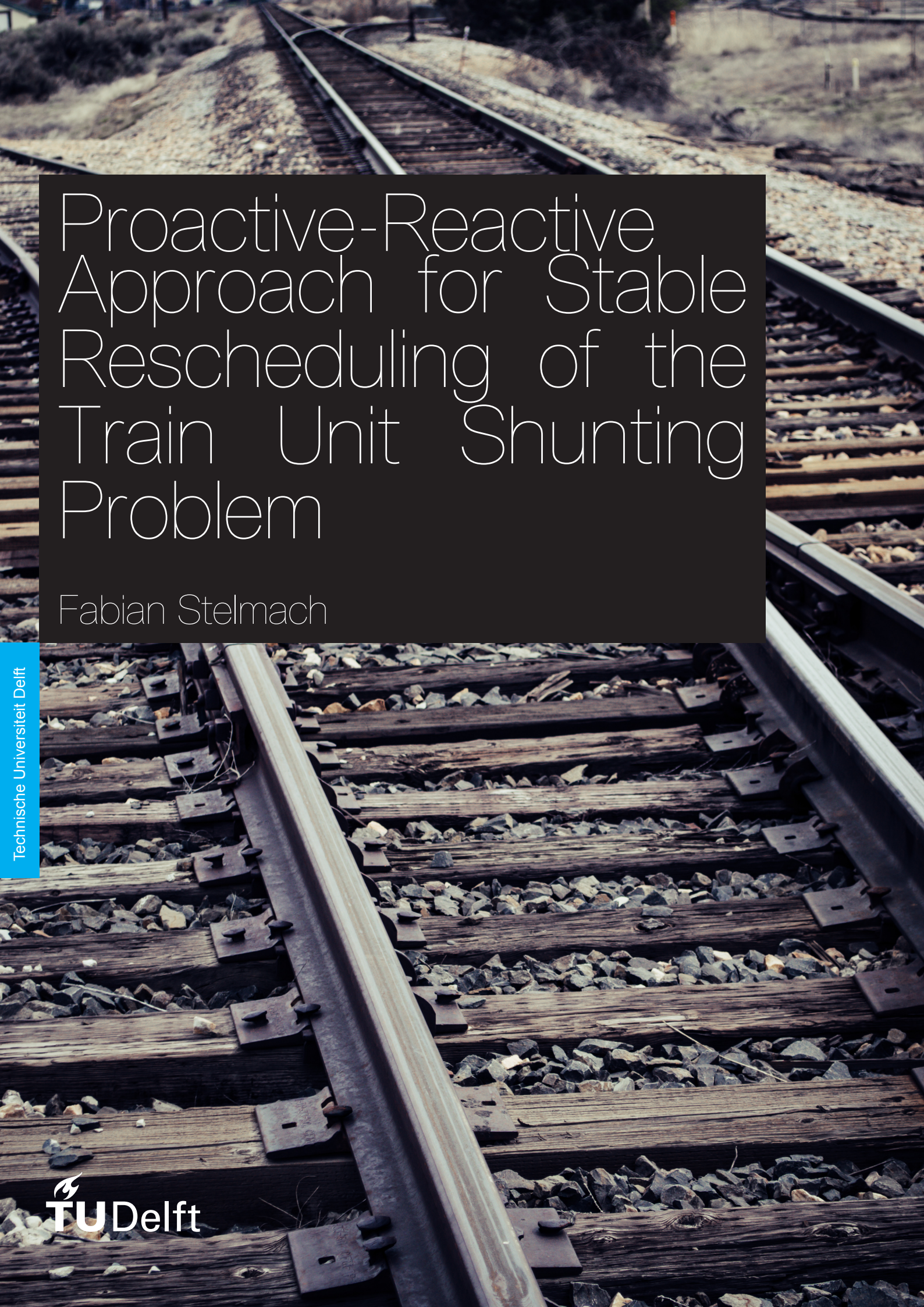

Proactive-Reactive Approach for Stable Rescheduling of the Train Unit Shunting Problem

Fabian Stelmach

Technische Universiteit Delft

TU Delft

Proactive-Reactive Approach for Stable Rescheduling of the Train Unit Shunting Problem

by

Fabian Stelmach

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Thursday August 13, 2020 at 14:00 PM.

Student number:	4291557	
Project duration:	12 November, 2019 – August 1, 2020	
Thesis committee:	Dr. Mathijs de Weerdt,	TU Delft, supervisor
	Prof. Karen Aardal,	TU Delft
	Dr. Matthijs Spaan,	TU Delft
	Dr. Laurens Bliek,	TU Delft

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft

Preface

This thesis has been written as partial fulfilment of the requirements of the degree of Master of Science in Computer Science at the Delft University of Technology. It is also the last requirement that I need to fulfil in order to complete the study of Computer Science at the Delft University of Technology, and as such, it marks the final stage of my academic career.

The topic of the thesis was chosen due to previous experience and academic curiosity. I have worked on various different projects regarding the scheduling of maintenance of trains during my academic career, in both the Bachelor as well as Masters the programme. Furthermore, I was interested in the efficiency at the dutch railways, as traveling by train in The Netherlands has proven to be a much more pleasant experience compared to my country of origin.

This thesis is the product of work made between November 2019 and August 2020. The project has been executed in collaboration with the Nederlandse Spoorwegen, as part of an internship.

I would like to thank my supervisors, Mathijs de Weerdt and Laurens Bliek for their excellent guidance. During the project, they have provided valuable support, feedback and guidance. I would not have been able to finish this project without their cooperation.

I also wish to thank Bob Huisman for his guidance as well. His professional insights have been invaluable for the understanding and excecution of the project. I also wish to thank Bob Huisman for making the internship at the Nederlandse Spoorwegen possible. The internship was helpful due to collaboration with my collageues at the Nederlandse Spoorwegen, and for this, I wish to thank them as well.

Finally, I would like to thank you, the reader, for spending your time on reading my work. I hope you enjoy the rest of this thesis.

Fabian Stelmach
Delft, July 2020

Abstract

This research is focused on the proactive-reactive rescheduling process of the Train Unit Shunting Problem (TUSP) on train maintenance shunting yards. An important difference between a scheduling process and the rescheduling process is that a reschedule must be both feasible and desirable (similar to pre-schedule), while a schedule does not require desirability. Firstly, an abstract proactive-reactive rescheduling framework is proposed. This framework is proposed as a base for the general problem of proactive-reactive rescheduling. A total of four reactive rescheduling methods for the TUSP are proposed and implemented as extensions to the Simulated Annealing local search algorithm. The extensions are created in order to achieve reschedule desirability by either guiding the local search process during its iterations or influencing its starting point. Using experiments based on realistic data, it is concluded that the Simulated Annealing can be used to create reschedules that are both feasible as well as desirable. It is shown that both proposed extensions are required for the best reschedule quality. Finally, further analysis of the results shows that the schedule resilience to late arrival of trains can be improved upon by increasing the amount of time between a train arrival and the time at which its services are scheduled.

Contents

1 Introduction

The train fleet at the The Dutch Railways (Nederlandse Spoorwegen, NS) requires regular maintenance. Such maintenance is performed on a shunting yard. During a 24 hour schedule horizon, different trains in various compositions arrive on the yard. Then, maintenance services are performed on some of the trains.

The maintenance schedule is usually created ahead of time. Various types of disturbances can occur during the execution of a maintenance schedule. The effect of a disturbance on the maintenance schedule varies greatly with the actual properties of the disturbances. In general, small disturbances can be dealt with by the maintenance staff. Larger disturbances may require significant changes to the maintenance schedule. Dealing with such disturbances requires a proper strategy. In this paper, such strategy is proposed.

1.1. Train Unit Shunting Problem: Context

In the Train Unit Shunting problem (TUSP), a train is a collection of at least one train unit in a specific order. A train unit is an entity with a specified length and type.

The shunting yard consist of multiple tracks and switches. Trains can pass through or park on tracks. A train can park on a track only if the train is shorter than the track they intend to park on. Passing through switches is allowed, but parking is not. Trains can only move from one track to another, if there is a path of connected tracks and switches between the two tracks, and that path is not occupied by another train.

The shunting yard Kleine Binckhorst is visualized in Figure 1.1. In the figure, track are represented by solid and dashed black lines. The switch IDs are represented by the red numbers, and the track IDs are represented by the black text, encapsulated in black capsules. There are multiple resources for performing the maintenance tasks. Internal cleaning can be performed on tracks 61 and 62, and technical inspection can be performed on tracks 52 through 59.

The trains arrive on the shunting yard according to an arrival timetable, on tracks designated for arrivals. The arrival timetable specifies the time the train arrives, as well as the composition of the train, in terms of its train units.

Some arriving train units also require particular maintenance services, such as cleaning or inspection, to be performed on the shunting yard before their departure. The services can only be performed on tracks with a proper resource. For example, the cleaning task must be performed on tracks with a cleaning station.

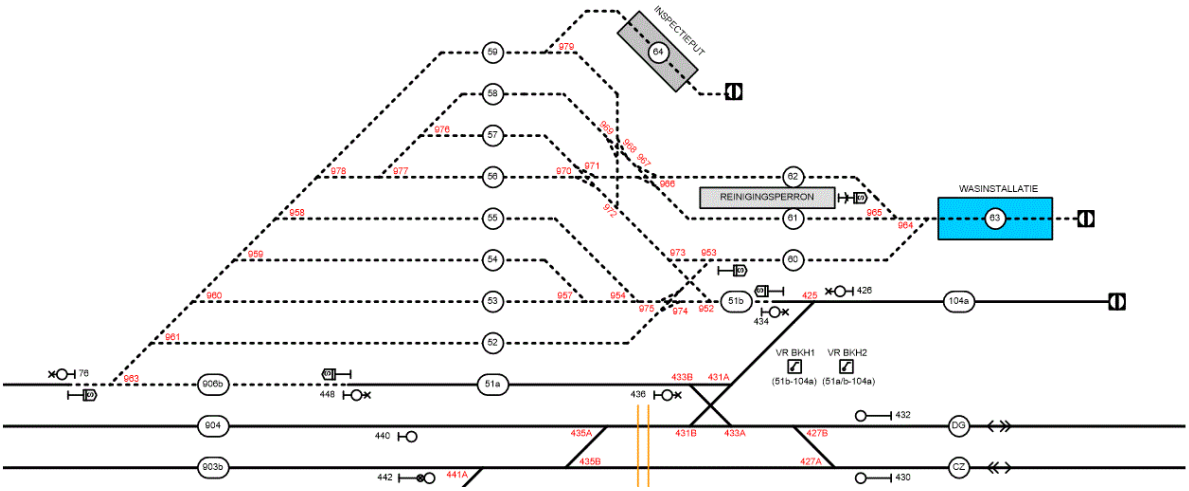

Figure 1.1: Kleine Binckhorst Shunting Yard. Retrieved from `http://www.sporenplan.nl/`[1]

The departure of trains is specified in the departure timetable. This timetable specifies the composition of trains and departure time. A train may depart on tracks designated for departure. Noticeably, there is not a fixed link between the train units in the arrival timetable and the departure timetable.

The collection of the information about the tracks, switches, the connections between tracks and switches, the length of the switches and the placement of maintenance resources is referred to as ”location”. The collection of information about the arriving and departing trains, together with their composition, length and required services is referred to as ”scenario”.

In the following subsections, The problem decomposition and the description of a solution to the problem are first presented in section 1.2. secondly, the uncertainties on the shunting yard and method for dealing with uncertainties are presented in section 1.3. Finally, the research questions are described in section 1.4.

1.2. Train Unit Shunting Problem: Solution

The solution to the TUSP can be divided into four different sub-problems: matching, task scheduling, parking and routing. In the following subsections, each of the four sub-problems is explained. A feasible solution to the TUSP is equivalent to a feasible solution to the four sub-problems. A solver for TUSP is a function that takes the location and scenario as input and produces a solution, if such a solution exists. A solution is viable if and only if it the solution of each of the four sub-problems in that solution is feasible.

1.2.1. Matching

Matching is the sub-problem of connecting the train units from the arrival timetable to the outgoing trains in the departure timetable. As such, matching can be visualized as a graph with a node for each incoming and outgoing train unit. For a matching solution to be feasible, every node must have a degree of 1. Furthermore, each edge connects an incoming train to an outgoing train of the same type.

An example matching is displayed in Figure 1.2. In this matching, the arriving train units are represented by blue nodes on the left, and the departing train units are represented by orange nodes on the right. The matching is feasible, since the degree of each node is 1, and each edge connects an arriving train and a departing train of the same type.

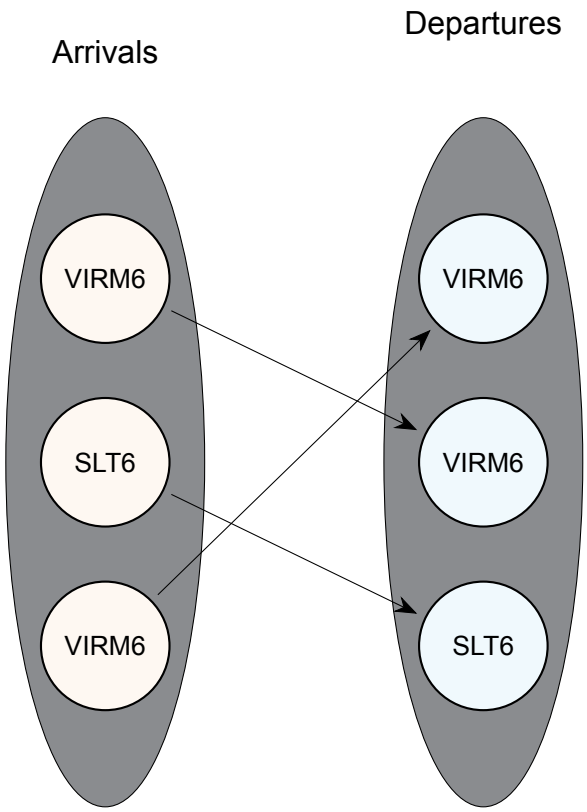

Figure 1.2: Example Matching

1.2.2. Task Scheduling

An arriving train unit might require some maintenance services (tasks) to be executed before departure. The list of required tasks for each arriving train is known ahead of time, before the arrival of the train. For the solution to be feasible, every service must be scheduled exactly once before the departure of the train. Furthermore, every service needs to be scheduled at a correct location, where the service

can be executed. Tasks can only be executed on tracks with proper resources. A solution to the task schedule sub-problem can be represented as sequence of tasks executed for each task. The sequence is ordered by the starting time of the tasks. Each task has a location, a starting time and an ending time. Only a single task can be executed on a single train unit at the same time. Only a single task can be executed on a given track at the same time.

An example task schedule is displayed in Figure 1.4. The schedule is feasible, since at no point do two task get scheduled on the same location, all the tasks are scheduled and no train unit is scheduled for two tasks at the same time. Note that two tasks of the same type can be scheduled in parallel, but only if they are executed on different tracks. Finally, there is usually a gap between two consecutive tasks on a given train unit. This is due to the fact that other actions, such as routing, may need to be performed in between the two consecutive service tasks.

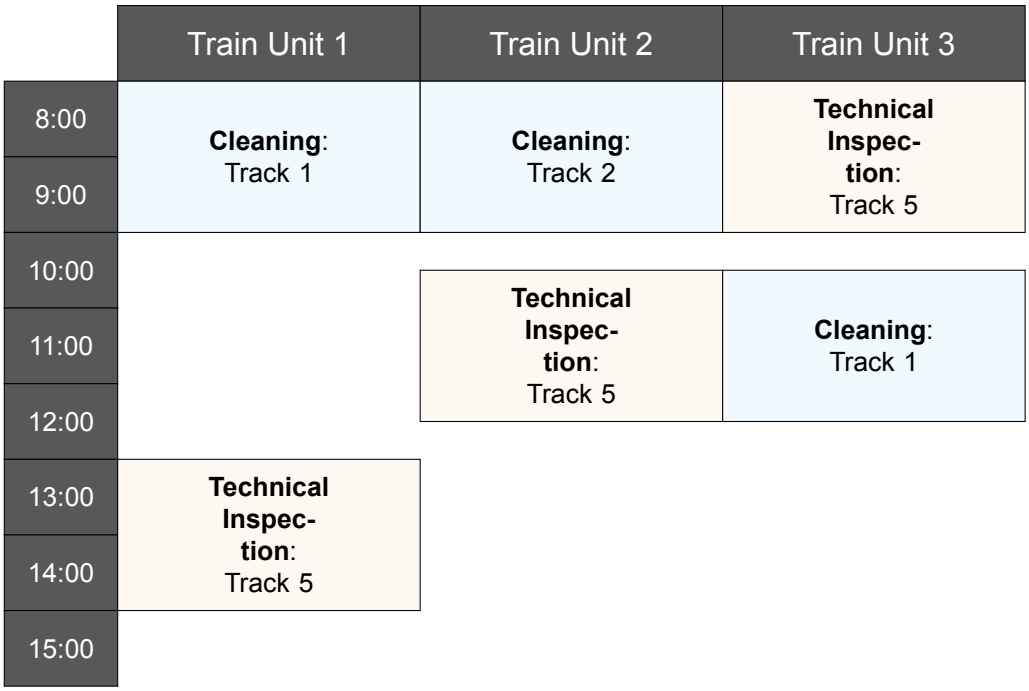

Figure 1.3: Example Routing

1.2.3. Parking

A train that has arrived to the shunting yard but did not depart yet always occupies some tracks at the shunting yard. This could be problematic, since the train could be occupying a track that is on the route of another train. The train should thus be parked on a track that, ideally, does not cause an obstruction to other trains. An additional constraint is enforced here, namely that a train parked on a given track might not exceed the length of the track.

1.2.4. Routing

After arriving on the shunting yard, a train might need to get transported to a service task, a parking job or the departure. The route from the location of the train to another location is a list of tracks and switches that are visited in order.

A feasible route must not be obstructed by other trains. Two different trains may occupy the same track, if the total length of the trains does not exceed the length of the track, and the trains do not cross each other. Furthermore, for every pair of tracks or switches (v_1, v_2) in a given route, if v_1 is directly followed by v_2, then v_1 must be connected to v_2. In other words, the train must be able to drive to v_2 from v_1 without visiting any other tracks or switches in between.

The routing is also dependant on the solution of the other four sub-problems. If a train has a service task scheduled at a given time, then that train must be present at the service location at the time when the task is scheduled. This is also true for train departure and parking.

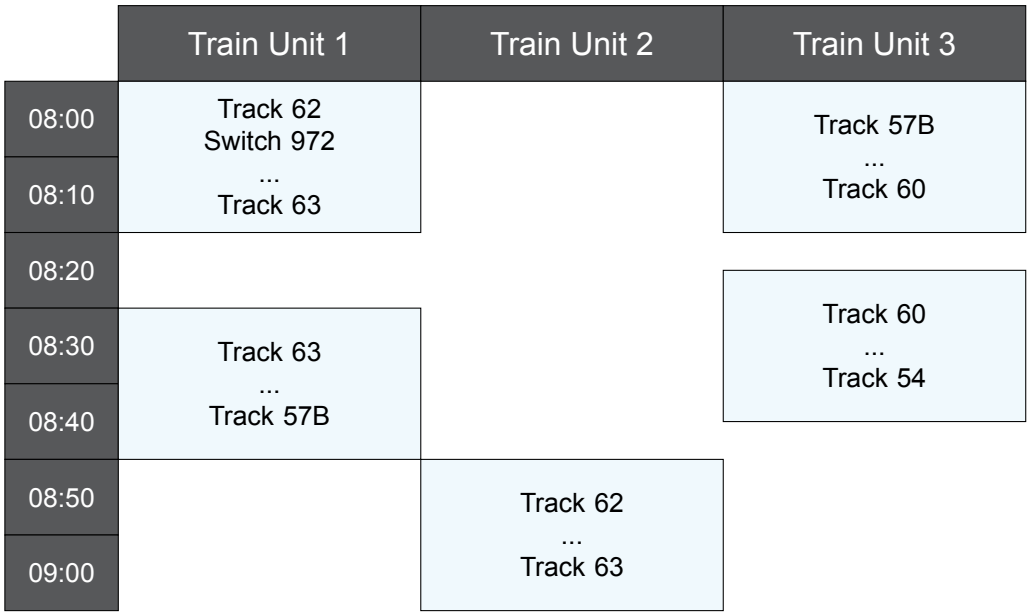

Figure 1.4: Example Task Schedule

1.3. Scheduling under Uncertainty

In traditional scheduling approaches, it is often the assumption that the plan and the execution of the plan are deterministic. The assumption is that there is little to no risk of tasks failing or being delayed when such plan is being executed. In practice, such disturbances do occur and they need to be dealt with. The uncertainty at the shunting yard is explained in subsection 1.3.1. The problem of dealing with disturbances is is more formally defined in subsection 1.3.2 The existing approaches for dealing with disturbances and the classification of such methods are explained in subsection 1.3.3.

1.3.1. Uncertainty at the Shunting Yard

The TUSP can be solved in a deterministic fashion. When such an approach is chosen, the assumption is made that every operation on the shunting yard can be executed at a specific time, within a predetermined time limit. Examples of such operations are arrivals of trains and execution of service tasks.

In practice, this is not the case. It is possible that the trains arrive late or in an unexpected composition. Furthermore, some service tasks may take longer to complete than expected. Figure 1.5 shows the deviations in train arrival times. In general, such deviations in time are called disturbances. A disturbance is a single change in the scenario. Disturbances may not be known at the time of the initial planning and may emerge during the execution of the schedule.

Figure 1.5 shows that trains do not always arrive at the scheduled times. While most trains arrive within 20 minutes of the scheduled arrival time, there are exceptions. In the figure, it can be seen that delays of 60 minutes can occur. For the sake of readability, Figure 1.5 only contains the data of trains that are up to 60 minutes late. The long tail of the curve is presented in Figure 1.6. In this figure, it can be seen that delays of multiple hours can occur, albeit less frequently.

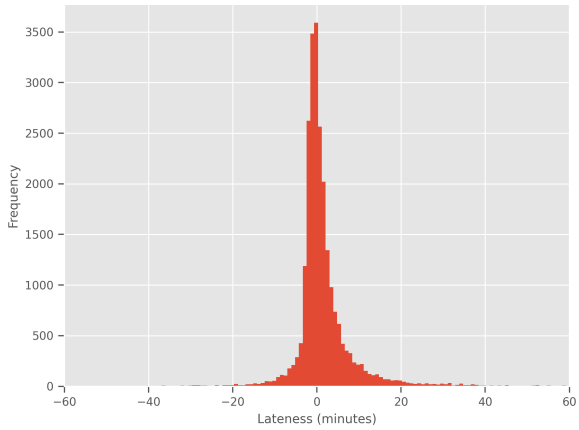

Figure 1.5: Lateness of Train Arrivals

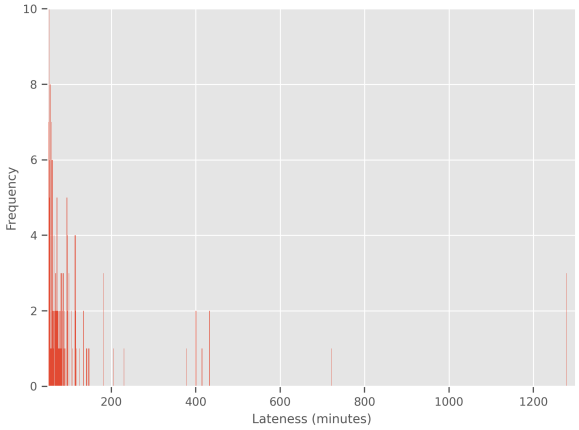

Figure 1.6: Long Tail of the Lateness of Train Arrivals

1.3.2. Rescheduling: Problem Definition

The pre-schedule is a feasible schedule for the initial, undisturbed problem instance. The reschedule, on the other hand, is a feasible schedule to the disturbed initial problem instance.

Secondly, due to the fact that the disturbances are not known ahead of time, the only parts of the pre-schedule that may change in the reschedule are those that happen after the presence of the disturbance is established. In other words, the actions in the past may not be changed. In general, the time at which the presence of the disturbance is established is called t_D.

Thirdly, the parts of the pre-schedule that happen after t_D may change in the reschedule, but they may only be rescheduled to a time later than t_D. This is due to the fact that the disturbance is not known before t_D, and therefore, rescheduling actions to a time before t_D would be equivalent to scheduling them in the past.

Finally, the reschedule should not deviate significantly from the pre-schedule. This property is generally referred to as stability. Note that there is not a single concrete definition of schedule stability. Stability is not necessarily required for the feasibility of the reschedule, but it is necessary for the acceptation of the reschedule, the desirability. The staff operating at the shunting yard requires the knowledge of the schedule ahead of time. Changes to the schedule require the staff members to study a new schedule in possibly short amount of time, and therefore, the number of changes should be minimized, or else the reschedule may lack desirability.

1.3.3. Scheduling Strategies

Disturbances may render a train schedule infeasible. When a solution is found with the assumption that the shunting yard is a deterministic system, there is usually no strategy defined for dealing with such disturbances. In order to deal with disturbances, some changes to the schedule need to be made.

The approaches are commonly evaluated in terms of the following properties: stability, flexibility. A scheduling approach is stable if the produced reschedule does not significantly change from the pre-schedule. A scheduling approach is flexible if it can produce a feasible schedule in the presence of various disturbances.

Scheduling under uncertainty can be approached in many different ways. In most classifications [5, 14] of the scheduling approaches under uncertainty, the scheduling approaches are characterized by two main properties: whether a pre-schedule is used and whether the schedule may be changed during its execution. The pre-schedule is equivalent to the initial schedule that is feasible for the undisturbed scenario. The schedule that is fixed for the disturbed scenario is called the reschedule. The classification of the most common scheduling strategies is displayed in Table 1.1.

Table 1.1: Scheduling Strategies

	No changes during execution	**Changes during execution**
No pre-schedule		Reactive Scheduling
Pre-schedule	Deterministic Scheduling	Predictive-Reactive Scheduling
Pre-schedule with Disturbance Anticipation	Proactive Scheduling	Proactive-Reactive Scheduling

Deterministic Scheduling In deterministic scheduling, the assumption is made that the execution of the schedule can be fully predicted ahead of time. There is no strategy present for dealing with disturbances. If a disturbance occurs, then the deterministic scheduling approach offers no solution for dealing with that disturbance.

Proactive Scheduling In proactive scheduling approach, the pre-schedule is usually generated in such a way that it can absorb disturbances that occur during the execution of the schedule. The tasks executed in the schedule are assumed to have stochastic start and finish times and the pre-schedule is generated such that in the expectation, the schedule can be executed successfully. This strategy does not allow the pre-schedule to be changed during the execution time. Some disturbances can not be handled by this scheduling approach, since the scheduling approach is not very flexible. An example of such disturbance is the addition of a new task to be executed in the schedule. Such disturbance can not be solved using just proactive scheduling. On the other hand, the proactive scheduling guarantees stability with respect to the pre-schedule, since changes to the pre-schedule are not permitted.

Reactive Scheduling In reactive scheduling, there is no notion of a pre-schedule. All the decision are made on-line, based on the state of the shunting yard. Reactive scheduling is more flexible than proactive scheduling, and so theoretically, more disturbances can be solved using this approach. The notion of stability does not exist, since there is no pre-schedule.

Predictive-Reactive Scheduling In the predictive-reactive scheduling, a pre-schedule is initially generated. When a disturbance occurs, the schedule is updated to accommodate for the disturbance. The reactive rescheduling step of this strategy may change the pre-schedule freely, and therefore, stability is not guaranteed with this approach.

Proactive-Reactive Scheduling The proactive-reactive scheduling method is an extension of the predictive-reactive scheduling method. Initially, a deterministic pre-schedule is generated. Then, after an occurrence of a disturbance, the pre-schedule is changed to accommodate for the disturbance. The pre-schedule generation differs from the predictive-reactive scheduling method in that the pre-schedule is made such that the rescheduling process performs better. There is a notion of anticipation of disturbances within the pre-schedule.

After having defined the various categories of possible rescheduling methods, it is clear that the proactive-reactive scheduling method is the most promising one to create stable reschedules. The implementation of this approach will be further studies in this paper.

1.4. Research Questions

Due to the uncertainty at the shunting yard, a solution for dealing with various disturbances is desirable. More specifically, trains arriving later than planned can cause a TUSP schedule to become infeasible. To deal with such disturbances, the proactive-reactive scheduling strategy may be applied. Therefore, in this paper, the following research question is answered:

Can proactive-reactive rescheduling method be used to deal with disturbed TUSP schedules?

In order to answer the research question, a number of sub-questions is formulated. In order to be able to apply the proactive-reactive scheduling method on TUSP, it is first necessary to specify what the components of a proactive-reactive scheduling algorithm are and how they are relevant for the problem. Therefore, the first proposed sub-question is:

What are the components of a proactive-reactive scheduling algorithm and how are they relevant for the TUSP?

One of the parts of the proactive-reactive scheduling method is the reactive part. For the reactive part of the proactive-reactive scheduling method, the simulated annealing local search TUSP solver[15] will be used. This solver is originally designed to be a scheduling algorithm and not a rescheduling algorithm. Therefore, an extension will be needed in order to allow the algorithm to solve rescheduling problems in a stable manner. For this reason, the third proposed sub-question is:

Is simulated annealing local search TUSP solver suitable to to create desirable reschedules?

Finally, the rescheduling performance can additionally be improved by changing the pre-schedule generation. The proactive pre-schedule generation algorithm should generate pre-schedules such that the rescheduling performance is increased. Most importantly, the rescheduling performance should be optimized with respect to schedule stability. Thus, the last proposed sub-question is:

Which property of the TUSP pre-schedules must be improved in order to improve temporal flexibility?

2

Literature review

The train unit shunting problem is well known in the literature. In order to find out to what extent the research questions can be answered by existing literature, a literature review is conducted. In this literature review, relevant papers regarding the proactive-reactive scheduling algorithm and the train unit shunting problem will be presented.

In section 2.1, the solving techniques for the train unit shunting problem are discussed. In section 2.2, the general techniques of robust scheduling and the implementations of them on the TUSP are explained. Finally, in section 2.3, the simulated annealing algorithm is explained.

2.1. Train Unit Shunting Problem

The train unit shunting problem is a well known problem in the literature. Multiple attempts have been made to create algorithms to solve the TUSP. The TUSP consist of four sub-problems, each of which is NP-hard [15].

The TUSP solvers can be divided into two main groups. Some solvers attempt to solve the TUSP by decomposing the problem into multiple sub-problems and then solve those sub-problems sequentially. Examples of works using this approach include the works by Freling et al.[4]. In this research, the TUSP is solved using the decomposition technique.

Other solvers take a different approach. A notable example is the HIP solver developed by Van Den Broek [15]. The approach is integrated, as the TUSP sub-problems are attempted to be solved concurrently, all at the same time. This solver is based on the Simulated Annealing local search technique.

2.2. Robust Scheduling

Van den Broek [16] and Kleine [8] have shown that there exist surrogate measures that correlate strongly with the robustness of the solutions to the TUSP. Both Van den Broek [16] and Kleine [8] measure robustness of the solutions in terms of the fraction of the schedule delayed and average tardiness of a train unit. Kleine [8] additionally used fraction of the tasks delayed and deviation from the (deterministic) earliest starting time as a robustness measure.

In their analysis, both Van den Broek [16] and Kleine [8] have used proactive-reactive scheduling approach. In case of deviations in the schedule, the execution of all tasks was delayed appropriately, in order to preserve the sequence order. This approach is called the right-shift reschedule approach.

Furthermore, Kleine [8] has proposed an adjustment to the integrated local search approach solver by

Van den Broek [15] in order to generate robust pre-schedules.

Finally, Bao [2] has proposed a robust scheduling approach based on decision trees. The robustness in this approach has been calculated by comparing the number of unique solutions generated to a set of specific, similar problem instances of TUSP. The decision trees based approach has generated the least unique solutions when compared to other solving techniques, making it the most robust one.

Leon et al. [9] has devised a proactive-reactive scheduling method for the job shop problem under random machine breakdown. The rescheduling algorithm used is the right-shift reschedule approach. This approach delays any action in the job shop until the execution can proceed, maintaining the order of the sequence in which the jobs were planned to be executed.

The use of right-shift reschedule approach causes absolute stability of the schedule when stability is defined in terms of the sequence of jobs executed. This it not the case when stability is defined in terms of the job starting times instead of sequence of the jobs. Other reschedule techniques may cause the realized schedule to differ from the pre-schedule both in terms of job starting times as well as the sequence of the jobs executed. The single-disruption stability problem with n parallel machines is strongly NP-hard. [11]

A bi-criterion reschedule algorithm under machine breakage in the context of one machine scheduling has been proposed by Wu et al.[19] The two criteria used are: efficiency and deviation from pre-schedule. Deviation between schedules is measured as either average absolute start time difference from the original schedule or the sequence changes from the original schedule or both. The problem is solved using a genetic algorithm.

2.3. Simulated Annealing

Simulated Annealing is a local search optimization technique. The general Simulated Annealing algorithm works by starting the optimization process with some initial solution and then iteratively improving the solution by exploiting the local neighborhood of the chosen solution [17].

The Simulated Annealing algorithm explores the solution space by modifying the initial solution minimally. The solutions which are easily reachable with a single step from a solution S are called the neighbors of S [3]. The neighbors of a solution are explored in order to find the best solution. The algorithm accepts a neighbor to be the new solution if the neighbor is a solution of higher quality than the current solution. If the neighbor is of lower quality, then it is accepted as a new solution in a stochastic manner [3]. The probability to accept a worse solution decreases over time.

The Simulated Annealing algorithm is further explained in the context of the TUSP in section 5.2.

3

Proactive-Reactive Scheduling Framework

The concept of proactive-reactive scheduling is well known in the literature, as previously described in subsection 1.3.3. In this rescheduling strategy, a pre-schedule is initially made using the proactive scheduling algorithm. After an occurrence of a disturbance, the pre-schedule is modified such that it is feasible with respect to the disturbed scenario. The pre-schedule is generated such that the schedule obtained after rescheduling performs better than other pre-schedules would. The performance is calculated with respect to an objective function.

Both the proactive as well as the reactive part of the rescheduling strategy are dependent on various factors of a rescheduling problem. An example of such factor is the disturbance model. An implementation of a proactive-reactive rescheduling strategy depends on the disturbance model that the strategy is applied on, and may not work with other disturbance models. The proactive-reactive rescheduling strategy depends on more factors than just the disturbance model. Examples of such factors are problem instance model, the rescheduling objective, the pre-schedule generation algorithm and the rescheduling algorithm. Those factors all influence the rescheduling process. To the best of our knowledge, a framework describing the factors and their role in the proactive-reactive rescheduling process has not been proposed yet. This makes it difficult to reason about and compare different proactive-reactive rescheduling strategy implementations.

In this chapter, an abstract framework describing the proactive-reactive pre-schedule rescheduling process is described. In section 3.1, the abstract general proactive-reactive rescheduling framework is proposed. In section 3.2 various implementations of the abstract framework are presented.

3.1. Proactive-Reactive Rescheduling Process

An implementation of a proactive-reactive strategy consists of two algorithms: a proactive schedule generation algorithm f_{pro} and a reactive schedule repair algorithm f_{re}. The two algorithms work on a specific problem domain. The domain dictates the disturbance model D, the problem instance model I and two objective functions: g_{pro} and g_{re}. The important distinction is that the choice of implementation of the two algorithms f_{pro} and f_{re} can be changed freely, while the choice of D, I and g is dictated by the problem domain.

In the event-driven proactive-reactive rescheduling strategy, a rescheduling problem is a problem in which a pre-schedule S_{pre} for a problem instance $i \in I$ is initially generated using f_{pro}. The pre-schedule is generated to optimize the function g_{pro}. Then, the problem instance $i \in I$ is disturbed with a disturbance $d \in D$. The pre-schedule S_{pre} is then modified in order to accommodate for d. The

solution to the disturbed problem instance i_D is the reschedule S_{re}. The reschedule is obtained using rescheduling algorithm f_{re}. The performance of the rescheduling algorithm is measured using g_{re}.

The components of the rescheduling process are described in more detail in the following subsections.

3.1.1. Problem Instance Model I

In general, scheduling problems are problems in which there is a set of jobs to be executed. Those jobs are mapped to a time slot, and possibly a resource, subject to some optimization. The problem instance model I describes the distribution over the possible problem instances $i \in I$. The problem instance i is a collection of information specific to the scheduling problem. It contains information about what jobs need to be executed in a feasible schedule and possibly problem specific constraints about those jobs. Furthermore, the available resources are also specified in the problem instance. Even though a single problem instance i is used in the process, the performance of the rescheduling process is generally computed with respect to I.

The problem instance may affect the performance of the rescheduling algorithm f_{re} with regards to the objective function g_{re}. This might not be helpful with regards to choosing the problem instance, since problem instances are usually given and need to be solved; there might not be any flexibility in choosing the problem instances. On the other hand, the knowledge that an incoming problem instance is difficult to reschedule may still be useful.

3.1.2. Disturbance Model D

The disturbance model D is a distribution of all possible disturbances that can occur in a given problem instance i. The disturbance $d \in D$ is the description in how the disturbed problem instance I_D differs from the undisturbed problem instance I.

There are many possible disturbance models and they may be problem specific. In general, a disturbance affects either the jobs to be executed or the available resources. While it is out of the scope of this thesis to create a comprehensive list of possible disturbance types, a few examples based on the study of Vieira et al.[18] are presented. Examples of disturbances affecting the jobs are[18]: due date change, process time change and job cancellation. Examples of jobs affecting the available resources are[18]: machine failure material arrival delay and shortage of materials.

A disturbance model is not limited to a single instance of the disturbance types described above, it may consist of a combination of multiple disturbance types. Furthermore, there is a lot of flexibility in defining the disturbance model.

A pre-schedule generated in a proactive approach performs well with respect to some specific disturbance model, and might not perform well with other disturbance models.

3.1.3. Objective Functions g_{pro} and g_{re}

The two objective functions g_{pro} and g_{re} are used to evaluate the quality of the schedules generated by f_{pro} and f_{re} respectively. The reschedule S_{re} is generated such that it optimizes f_{re} the pre-schedule S_{pre} is generated such that S_{re} optimizes f_{pro}.

The function g_{pro} is the objective function that the proactive approach attempts to optimize. This objective function contains information about the performance of the reactive scheduling process. More specifically, it describes the in which aspect of the scheduling problem, the reactive process will perform better when a pre-schedule is generated using the proactive process.

The objective function in the proactive approach may be the same as the objective function g in the rescheduling approach, but it is not necessary. The objective function of the proactive approach may, for example, consist of a single objective, when the objective function of the reactive approach consists of multiple objectives.

The objective specification for the proactive scheduling process is also common in literature. In their survey about project scheduling under uncertainty, Herroelen et al. [6] identify three different objectives which a pre-schedule can be optimized for. In this research, the objectives for the proactive scheduling process closely follow those specified by Herroelen et al.

The first possible objective is the schedule efficiency. In literature, this is often called *quality robustness*[6, 8, 16]. A pre-schedule that is made specifically for maintaining schedule efficiency, will result, after rescheduling, in a reschedule that is similar to the pre-schedule in terms of the objective value of the scheduling problem.

The second possible objective is the schedule similarity. In literature, this is often called *solution robustness* [8, 16] or *stability* [6]. A pre-schedule that is made for maintaining schedule similarity, will be rescheduled into a reschedule that is similar to the pre-schedule in the solution space.

The third objective is the schedule flexibility. A pre-schedule is flexible, if a feasible reschedule can be calculated for a large range of possible disturbances. Notice the dependency on the disturbance model used. A pre-schedule can only be flexible with regards to some specific disturbance model.

Even though only three possible objectives are specified, there is a lot of flexibility within the choice of the objective. Firstly, a pre-schedule can be optimized for a combination of multiple objectives. Furthermore, each of the objectives can be defined in various ways. The schedule efficiency is often defined in terms of the schedule makespan, but it can also be defined in other terms, such as required resources. In general, schedule efficiency, schedule similarity and flexibility are terms which require more specific definitions that often depend on the scheduling problem.

The second objective function g_{re} can be defined in the same terms as g_{pro}. It is often the case that g_{pro} is equal to g_{re}, but this is not required. It is possible, for example, for g_{pro} to include less measures than g_{re}. In such cases, the rescheduling algorithm f_{re} optimizes S_{re} for multiple objectives, and the choice in S_{pre} increases performance for a subset of those objectives.

3.1.4. Pre-schedule Generation Algorithm f_{pro}

The pre-schedule generation algorithm f_{pro} is used to generate pre-schedules for problem instances $i \in I$. The pre-schedule S_{pre} is a feasible solution to the undisturbed problem instance i. The pre-schedule contains the mapping from jobs of the problem instance to time slots and (possibly) resources. The choice of pre-schedule generation algorithm can affect the performance of the rescheduling algorithm.

The proactive pre-schedule generation algorithm is not limited to a single solution method, but the most prevalent one is the use of surrogate measures[8, 10, 16] that correlate to the objective function g_{pro}. A commonly used technique to increase the rescheduling performance is to increase the amount of slack in the generated pre-schedules. Schedules with more slack have been proven to be more resistant to temporal disturbances than schedules with less slack.

3.1.5. Rescheduling Algorithm f_{re}

The rescheduling algorithm modifies the pre-schedule in order to accommodate for some disturbance $d \in D$. Two important characteristic factors of a given rescheduling algorithm are: the repair method used and the rescheduling point. Differentiation between rescheduling algorithm based on their repair method or the rescheduling is common in literature, and therefore, the both the factors is discussed here.

The repair method is the allowed set of rules that may be applied on S_{pre} in order to create S_{re} that is feasible with respect to i_D.

The repair method is used in the reactive step to repair the pre-schedule after a disturbance has occurred. While the repair method is used only in the reactive step, the choice of the repair method is significant for the pre-schedule generation. A pre-schedule created in a proactive approach is targeted

to a specific repair method and may not necessarily work with other repair methods. A pre-schedule generated for the right shift rescheduling method might not perform well when used with full rescheduling method.

Commonly occurring examples of repair methods are: right shift rescheduling, partial rescheduling, full rescheduling and rescheduling with instance relaxation.

- Right shift reschedule: only allowing the jobs to be delayed, but disallowing the change in the relative order of execution of the jobs.
- Full rescheduling: allowing changes in both the time slots of the jobs as well as the relative execution order and the mapping to the resources.
- Partial rescheduling: allowing changes similar to those of full rescheduling, but the rescheduling point is set to after the occurrence of a disturbance. Any actions before the rescheduling point may not change.
- Rescheduling with instance relaxation: in this method, the problem instance itself may be changed (relaxed) to make the rescheduling process easier. Such changes include: increasing the resource capacity or dropping jobs partially or fully.

The distinction for proactive schedule generation depending on the repair method is common in rescheduling literature. An example of this is the distinction between robust and flexible pre-schedules.

On one hand, a robust pre-schedule is a schedule, that after an occurrence of a disturbance, performs better than ordinary schedules when rescheduled using the right shift reschedule repair method [7, 8, 16, 18]. On the other hand, a flexible pre-schedule performs better than ordinary schedules when rescheduled using a repair method other than simple rescheduling methods such as right shift reschedule [18]. The difference between a robust pre-schedule and a flexible pre-schedule is the repair method for which the pre-schedule is designed for. This is an example of the generalization that the proactive pre-schedule generation process is dependant on the repair method of the rescheduling algorithm.

The choice of the repair method is linked to the disturbance model D. Not all disturbance types can be solved using every repair method. As an example: the right shift reschedule can not be used to solve trains arriving in unexpected order. This disturbance model causes changes to the shunting yard state that require schedule changes that the right shift reschedule is not able to provide.

While the choice of the rescheduling point is less commonly discussed in literature, it is an important characteristic of the rescheduling algorithm. The rescheduling point is the point in time at which the rescheduling takes place. No actions can be changed before the rescheduling point. Commonly used values for the rescheduling point are: time of the occurrence of the disturbance and rescheduling point ahead of time.

In the case where the rescheduling point is at the time of the occurrence of the disturbance, it is often the assumption that the disturbance was not known until it has occurred. Some disturbances are known ahead of time, but after the creation of a pre-schedule. In such cases, the rescheduling point may also be before the execution of the pre-schedule.

3.2. Implementation of the Framework

The framework provides an abstract way of representing proactive-reactive rescheduling process. In this chapter, the existing work regarding the proactive-reactive rescheduling of the TUSP explained using the framework. To the best of our knowledge, a total of three rescheduling techniques have been created for the TUSP. The surrogate based rescheduling methods are presented in subsection 3.2.1 and the decision based method is explained in subsection 3.2.2. Finally, in the following chapters of this paper, an implementation of the proactive-reactive process is described. In subsection 3.2.3, this implementation is explained in terms of the framework. This section is also summarized in Table 3.1.

3.2.1. Surrogate Measure Based Methods

In their work, Van Den Broek [16] and Kleine [8] have proposed similar, surrogate measure based methods for dealing with disturbances.

In their work, both Van Den Broek as well as Kleine propose a method for dealing with temporal disturbances of both train arrivals as well as task durations. Noticeably, both of the proposed methods only deal with relatively small temporal disturbances. The disturbances are small in the sense that the relative order of tasks and arrivals occurring is preserved despite the disturbances occurring. Both of the works are focused on realistic TUSP instances on the Kleine Binckhorst shunting yard, and the method by Van Den Broek is also applied on the Utrecht shunting yard.

Both the proposed methods focus on the efficiency of the schedule and temporal similarity in both the proactive as well as the reactive scheduling parts.

Both Van Den Broek and Kleine have proposed to guide the pre-schedule generation using surrogate measures. The surrogate measures correlate with the reschedule efficiency and temporal similarity. While Kleine has extended the work of Van Den Broek by including more possible surrogate measures, their approach to pre-schedule generation was fundamentally similar.

Finally, both the authors have worked with the right shift rescheduling algorithm. This is possibly the primary reason why only small temporal disturbances were considered, as right shift reschedule does not offer enough flexibility for the handling of large temporal disturbances. If, for example, a train is so late that the relative arrival order of the trains change, then right shift rescheduling will be unable to provide a solution for such scenario. The rescheduling point of both the approaches is equal to the time of the occurrence of the disturbance.

3.2.2. Decision Trees Based Method

Bao [2] has devised a method for rescheduling the TUSP using decision trees. In their approach, Bao uses small temporal differences in the train arrival times. Similarly to the surrogate measure based methods described above, the temporal differences are small enough not to disturb the relative order of the train arrivals. The disturbances are applied on artificial TUSP instances on an artificial shunting yard.

The objective for both the proactive as well as the reactive part is the similarity measure. Moreover, similarity described in the works of Bao relates only to the routing of the trains at the yard.

Both the pre-scheduling algorithm as well as the rescheduling algorithm is based on decision trees. The decision trees classifier is used to create schedules and repair them. The classifier takes similar decisions when creating the pre-schedule as well as when rescheduling. This results in schedule stability.

3.2.3. Local Search Based Method

In this paper, the main focus with regards to the disturbance model is on the variability of time arrivals. Contrary to the rest of the approaches described above, the focus is on large disturbances. The disturbances are large in the sense that after an occurrence of a disturbance, the relative arrival order of

trains is disrupted. The method is devised to work on realistic TUSP instances on the Kleine Binckhorst shunting yard.

On one hand, the objective function for the reactive part is defined in terms of efficiency as well as routing, temporal and spatial similarity. On the other hand, in the proactive scheduling part, the objective is to increase the rescheduling performance with regards to just the temporal similarity.

The proposed rescheduling algorithm is based on local search algorithm. This approach is flexible enough to allow for rescheduling of large disturbances. No concrete implementation of pre-schedule generation is proposed. Instead, pre-schedule characteristics that cause the lack of temporal flexibility are described.

In this chapter, an abstract framework for the proactive-reactive rescheduling strategy has been proposed. This framework describes the elements necessary to form a proper proactive-reactive rescheduling strategy. This framework will be used throughout the paper to describe the elements of the rescheduling strategy that will be proposed in the following chapters.

Table 3.1: Proactive-reactive Scheduling Methods for the TUSP

Author	**Problem Instance Model**	**Disturbance Model**	**Proactive Part Objective Function**	**Reactive Part Objective Function**	**Pre-Schedule Generation Algorithm**	**Rescheduling Algorithm**
Van Den Broek	Realistic TUSP instances on Kleine Binckhorst and Utrecht shunting yards	Small Time Variability of Arrivals and Tasks	Efficiency Temporal Similarity	Efficiency Temporal Similarity	Local Search using Surrogate Measures	Right Shift Rescheduling
Kleine	Realistic TUSP instances on Kleine Binckhorst shunting yard	Small Time Variability of Arrivals and Tasks	Efficiency Temporal Similarity	Efficiency Temporal Similarity	Local Search using Surrogate Measures	Right Shift Rescheduling
Bao	Small TUSP instances on artificial shunting yard	Small Time Variability of Arrivals	Routing Similarity	Routing Similarity	Decision Trees	Partial Reschedule with Decision Trees
Stelmach	Realistic TUSP instances on Kleine Binckhorst shunting yard	Large Time Variability of Arrivals	Temporal Similarity	Routing, Temporal and Spatial Similarity Efficiency		Partial Rescheduling with Local Search

4

Train Unit Shunting Problem Schedule Desirability

A suitable reschedule to the train unit shunting problem is characterized not only by its feasibility with respect to the disturbed problem instance, but also its similarity to the pre-schedule. In this chapter, the notion of schedule desirability will be expanded. The goal is to define a function which can be used to determine how desirable a reschedule is, given some pre-schedule. More formally, the goal is to define the objective function g_{re}, as defined in subsection 3.1.3. The function g_{re} is used to evaluate the quality of schedules generated by the rescheduling algorithm f_{re}. The quality is mostly optimized for schedule desirability.

In order to define the criteria of a desirable reschedule, the effects of a rescheduling must first be discussed. Many people with various professions are involved in the execution of the maintenance tasks on the shunting yard. The changes in the execution of the schedule may disrupt the workflow of the crew or incur other costs. For this reason, changes to the maintenance schedule are generally unwanted. In this chapter, a method to calculate the desirability of a TUSP reschedule is devised. In section 4.1, the effects of the reschedule and the criteria for schedule similarity are explained. Then, in section 4.2, the method for calculating TUSP reschedule desirability is proposed.

4.1. Motivation

One of the important actors involved in the maintenance schedule execution is the human planner. Human planners are involved with both the scheduling as well as the rescheduling process, even if both of the processes are partially automated using rescheduling algorithms. The planners create solutions to the rescheduling problems, either with their own knowledge or by using rescheduling algorithms. In certain situations, the planners are required to make quick decisions when there is no time to consult the algorithm. If the schedule is minimally changed by the rescheduling algorithm throughout the execution of the schedule, then the manual rescheduling process is made easier, as the human planners might not able to study a reschedule that significantly deviates from the pre-schedule quickly enough. The planners thus prefer the schedule to change as little as possible.

The second party involved with the execution of the maintenance schedule is the maintenance crew. The maintenance crew is responsible for executing the maintenance task such as cleaning and repair of trains on the shunting yard. In order to fulfill their task, the maintenance crew need to be at a certain service station at a given time. On most shunting yards, a single type of maintenance tasks can be executed on multiple servicing stations. The maintenance crew members must travel to different service stations, where the trains are scheduled for maintenance. The maintenance crew gets to know their schedule several hours in advance. A schedule of a single maintenance crew member contains

information about the scheduled location and starting time of each service task to be executed by that crew member. Changes to the location or starting time of service tasks disturb this schedule and thus, they should be avoided.

Finally, on some shunting yards, the route of serviced train units must be programmed before the execution of the maintenance schedule. The route of a train unit is an ordered sequence of train tracks and switches that the train unit passes through, starting at the arrival at the shunting yard and ending on the departure. The route is programmed beforehand in order to automate the operation of switches and to ensure safety. Changes to train unit routes are costly, since the changes need to be propagated to the controller systems at the shunting yard. Noticeably, the changes to the route of a single train unit are very costly and should be avoided, but changes to the execution order of movements of two different train units are not costly, since they can be programmed easily.

4.2. Dissimilarity Function

A reschedule should be not only be feasible, but also desirable. For a reschedule to be desirable, it must be similar to the pre-schedule. Large changes to the schedule due to disruptions should be avoided, since they disrupt the workflow of the maintenance crew and the human planners. Furthermore, changes in schedule can be costly to implement during the execution of the maintenance schedule.

In order to create similar schedules, the notion of similarity will first be defined. For calculating the similarity between two TUSP schedules, a dissimilarity function is used. The dissimilarity function is a function that takes as input two TUSP schedules (domain S) and outputs a non-negative, real number:

$$d : (S, S) \rightarrow \mathbb{R}_{\geq 0} \tag{4.1}$$

The dissimilarity function is a measure of relative difference between two TUSP schedules. The dissimilarity function evaluates to 0 if for two schedules S_i and S_j, the two schedules S_i and S_j are similar.

$$d(S_i, S_j) = 0 \text{ if } S_i \text{ and } S_j \text{ are similar} \tag{4.2}$$

The function evaluates to a a nonzero value when two dissimilar schedules S_i and S_j are used as input:

$$d(S_i, S_j) > 0 \text{ if } S_i \text{ and } S_j \text{ are dissimilar} \tag{4.3}$$

Finally, the function can be used to compare the dissimilarity between two pairs of TUSP schedules.

$$d(S_i, S_j) > d(S_i, S_k) \text{ if } S_i \text{ is more dissimilar to } S_j \text{ than it is to } S_k \tag{4.4}$$

The dissimilarity function should reflect the perceived similarity of two schedules. If two schedules S_i and S_j are similar to each other, then the cost to execute S_j when S_i was originally planned should be minimal. The schedule dissimilarity function can be used to ensure desirability of reschedules. A desirable reschedule minimizes the schedule dissimilarity function d.

Such a function is not straight forward to implement, since the exact requirements with regards to the reschedule changes are not known. Furthermore, the measure of dissimilarity is inherently multi-objective, as the various actors affected by the rescheduling have varying measures for the similarity of the reschedule. An implementation of the dissimilarity function is presented in section 4.3.

4.3. Implementation

In order to reflect the perception of schedule dissimilarity in the schedule dissimilarity function, a compound dissimilarity function is proposed. The proposed function d consists of four sub-components, reflecting the similarity criteria described in section 4.1. The four sub-components are: task starting time dissimilarity d_t, task location dissimilarity d_l, task routing dissimilarity d_r and task matching dissimilarity d_m. The dissimilarity function d is the weighted sum of the four sub-components: d_t, d_l, d_r and d_m. The four sub-components are weighted by their respective weights: $w_{d_t}, w_{d_l}, w_{d_r}, w_{d_m}$.

$$\begin{aligned} d = \; & w_{d_t} * d_t \\ + \; & w_{d_l} * d_l \\ + \; & w_{d_r} * d_r \\ + \; & w_{d_m} * d_m \end{aligned} \tag{4.5}$$

The four sub-components are explained in detail in the following subsections.

4.3.1. Task Starting Time Dissimilarity d_t

The first sub-component of the dissimilarity function is the task starting time dissimilarity. Each maintenance task on the shunting yard is performed on a given service location during some scheduled time. The maintenance crew members prefer not to have their schedules changed, and as such, the time at which the tasks of the crew are executed should not change. Assuming the disturbance model based on late arrival of trains on the shunting yard, variability of the starting time of service tasks is expected and should be minimized.

For a task t and two schedules S_i and S_j, the starting time of t in S_i should be equal to the starting time of t in S_j. Notice that the task t can have different starting time or service location between S_i and S_j, but it is assumed that the service type and duration do not change. Furthermore, in the chosen disturbance model, it is assumed that no tasks are removed nor added between the pre-schedule and the reschedule. Therefore, the two schedules contain the same tasks, with possibly changed location and starting time. For these reasons, the dissimilarity function is defined as the sum of absolute lateness of tasks in the two schedules. Notice that due to the fact that the absolute value of the lateness is calculated, a schedule becomes more dissimilar with respect to the starting time when in the reschedule, the task is scheduled earlier or later than in the pre-schedule. The full equation is presented below.

$$d_t(S_i, S_j) = \frac{1}{E} \sum_{t \in tasks} |ST_i(t) - ST_j(t)| \tag{4.6}$$

The sum of absolute task latenesses is normalized with respect to the normalization constant E, in order to make the metric comparable over various problem instances. The normalization constant in the case of a single late train is defined as the train delay multiplied by the number of tasks that of the delayed train. Note that this does not include the arrival delay. With this normalization, the value of d_t for a reasonable reschedule is expected to be between 0 and 1. The reason for this is that for a schedule with single train unit with consecutive tasks without any time between them, the worst case rescheduling performance is exactly E. This is visualized in Figure 4.1.

In this example, we see that the single train is delayed by two hours. There are no other trains on the shunting yard. In both of the schedules, there is a single inspection task that has to be executed during the maintenance period. In the reschedule, the inspection task is executed 2 hour later than in the pre-schedule. Notice that the inspection task can not be delayed any less, since it can not occur before the arrival of the train. Since there is just a single task for the delayed train and the train is delayed by two hours, $E = 2 * 1 = 2$. The inspection task is also delayed by two hours, so $\sum_{t \in tasks} |ST_i(t) - ST_j(t)| = 2$. Thus, $d_t(S_{pre}, S_{re}) = 1$.

Notice that the delay of the service task is not always equal to the train delay. To demonstrate this, in Figure 4.2 an example rescheduling scenario is presented. In this scenario, there is a total of two service tasks for the single delayed train.

In this example, there is again just a single train on the shunting yard. The train has two tasks that need to be executed: an inspection task and a cleaning task. The train arrival is delayed by two hours, and there are two tasks to be executed, so $E = 2 * 2 = 4$.

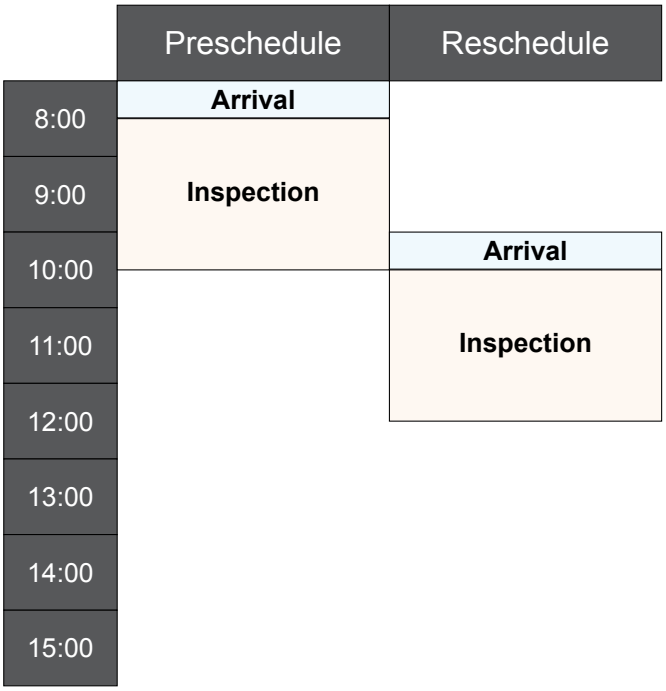

Figure 4.1: Worst Case Rescheduling Performance

Whereas the inspection task is delayed by two hours, the cleaning task is not delayed at all. The cleaning task occurs at the same time in both the pre-schedule as well as in the reschedule. Therefore, the sum of total latenesses in this situation is 2. Therefore, $d_t(S_{pre}, S_{re}) = \frac{2}{4} = \frac{1}{2}$. Intuitively, the rescheduling is more successful in this example than it was in the example in Figure 4.1, since only half of the tasks have been delayed in this example, as opposed to all the tasks in the example in Figure 4.1.

In both the presented examples, the schedule of just a single train unit is displayed. If more train units are present, then this calculation is applied to all the service tasks of those train units.

Notice that d_t is a proper dissimilarity function. For two schedules S_i, S_j, $d_t(S_i, S_j) \geq 0$, since d_t is a sum over latenesses. Furthermore, $d_t(S_i, S_j) = 0$ only if the service tasks are scheduled at the same times in both S_i and S_j. If the service tasks are not scheduled at the same time in S_i and S_j, then $d_t(S_i, S_j) > 0$. Finally, $d_t(S_i, S_j)$ becomes larger if the sum of latenesses becomes larger, so that a larger d_t is equivalent to a larger dissimilarity between the two schedules.

4.3.2. Task Location Dissimilarity

Similarly to the task starting time dissimilarity, the location of the service tasks is important for the maintenance crew members, since deviations in the service location can disrupt their personal schedules. The task location may vary between the pre-schedule and the reschedule. The location of a service task may be changed in order to repair the schedule after an occurrence of a disturbance.

For a task t and two schedules S_i and S_j, there exists a difference in task location if t is executed on a different location in S_i than it is in S_j. The task location dissimilarity is defined as the normalized count of tasks that have a different location between the schedules S_i and S_j. The task location dissimilarity is normalized with respect to the number of tasks present in the schedules. The measure of location difference for a single task is binary, either the location between two schedules changes in S_j as compared to S_i, or it does not. The assumption is that each task location change has the same cost assigned to it.

Let $loc_k(t)$ denote the location of task t in schedule S_k, then the task location dissimilarity function is

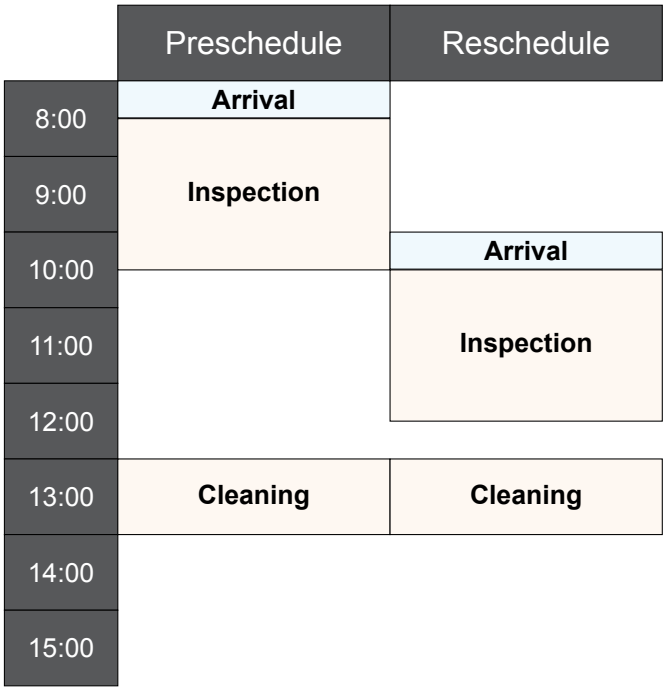

Figure 4.2: Example Maintenance Task Reschedule

defined as follows.

$$d_l(S_i, S_j) = \frac{1}{|tasks|}\left|\{t \in tasks | loc_i(t) \neq loc_j(t)\}\right| \tag{4.7}$$

The location dissimilarity function is a function $d_l : (S, S) \to [0, 1]$, where values close to 1 represent high dissimilarity and values close to 0 represent high similarity between the two schedules. The function is undefined when there are no tasks to be executed, so the assumption is that $|tasks| > 0$.

The dissimilarity function will be illustrated using an example in Table 4.1.

Table 4.1: Example task location dissimilarity

	Cleaning 1	**Cleaning 2**	**Cleaning 3**	**Inspection 1**	**Inspection 2**
$\mathbf{S_i}$	Track 1	Track 2	Track 2	Track 2	Track 1
$\mathbf{S_j}$	Track 1	Track 2	Track 1	Track 2	Track 2

In this example, a total of 5 service tasks are presented. Out of the 5 tasks, only the tasks *Cleaning 3* and *Inspection 2* differ in the task service location between S_i and S_j. There is a total of 5 tasks and 2 tasks have their location changed, and thus, $d_l(S_i, S_j) = \frac{2}{5}$.

The dissimilarity function d_l is a proper dissimilarity function. As mentioned before, the function has range of $[0, 1]$. The function d_l evaluates to 0 if and only if the location of the service task is unchanged between the two schedules. If any of the location is changed, then $d_l > 0$. The value of d_l is larger when there are more task location mismatches, so d_l is a measure of dissimilarity.

4.3.3. Routing Similarity

The reprogramming of a train unit route due to rescheduling can be expensive on some shunting yards. The route of each train unit should be changed as little as possible, in order to keep the reprogramming costs low. The reprogramming costs are incurred when a route of a train unit changes. The costs are

dominated by the changes of routes of trains, not the total ordering of movements made by different train units.

A route of a train unit is a a list of tracks and switches that are visited by a given train unit, ordered according to the visit order. The route of a train unit can be changed in three distinct ways: a node in the route can be added, removed or replaced by another node. It is assumed that a route in the reschedule is obtained by the route of the same train in the pre-schedule by applying the three route operations consecutively. The edit distance of the route is then the least number of operations required to transform the route from the pre-schedule to the route from the reschedule. This edit distance is generally known as the Levenshtein Distance[12]. The Levenshtein Distance can be calculated in $O(|route_i| * |route_j|)$ using dynamic programming algorithms[13].

Let $lev(route_i(TU_k), route_j(TU_k))$ denote the Levenshtein Distance between the route of train unit TU_k in schedules S_i and S_j. To reflect the properties of the incurred costs due to reprogramming, the following routing dissimilarity function is proposed.

$$d_r(S_i, S_j) = \frac{1}{|TU|} \sum_{TU_k \in TU} \frac{lev(route_i(TU_k), route_j(TU_k))}{max(|route_i(TU_k)|, |route_j(TU_k)|)} \tag{4.8}$$

The routing dissimilarity measure is normalized with respect to the number of train units as well as the train route length. In the latter normalization, the useful fact about Levenshtein Distance is used, namely that the upper bound of the Levenshtein Distance is the length of the longest input sequence.

To illustrate this dissimilarity function, the example in Table 4.2 is used. In this example, the original route of a single train unit is transformed into the destination route in 3 steps. The original and the destination routes are in the first, respectively last rows of the table. The route starts at the track on the leftmost column and ends on the track on the rightmost column. The intermediate routes used for calculation of the edit distance are present in rows 2, 3 and 4. For the intermediate routes, the change from the previous route is marked blue and denoted in the leftmost column. Note that the empty table cells are used for alignment of the presented routes and should simply be ignored when reading the route.

Table 4.2: Example Routing Dissimilarity

Source Route	**Track 1**	**Track 2**	**Track 3**		**Track 5**	**Track 6**	**Track 7**
Add track 8	Track 1	Track 2	Track 3	Track 8	Track 5	Track 6	Track 7
Remove track 2	Track 1		Track 3	Track 8	Track 5	Track 6	Track 7
Replace track 5 with track 9	Track 1		Track 3	Track 8	Track 9	Track 6	Track 7
Destination Route	**Track 1**		**Track 3**	**Track 8**	**Track 9**	**Track 6**	**Track 7**

The Levenshtein Distance between the source and destination routes is 3. Both the source and destination routes have length of 6. The dissimilarity function is calculated for a single train unit. Therefore, the following holds: $d_r(S_i, S_j) = \frac{3}{6} = \frac{1}{2}$.

The routing dissimilarity function is a proper dissimilarity function. Due to normalization, the range of the function is $(0, 1)$. The function is 0 only if all the train unit routes are the same in the two compared schedules. The function evaluates to a value larger than 0 if there is at least one difference in the routing of the train units. Finally, more changes in the routing lead to a larger values of d_r.

4.3.4. Matching Similarity

The final dissimilarity metric is the matching dissimilarity metric. The matching changes should be minimized in order to reduce the perceived rescheduling dissimilarity, as changes in matching result in

a different train unit departure time.

The matching of a schedule is a mapping from the incoming train units to the departing train units. In order to compare the matching changes, the changes in the mapping are calculated. Let e denote an edge between arriving train unit to a departing train unit in the matching graph, and $matching_k$ denote the edges of the matching graph of schedule S_k. Then, d_m can be expressed as follows.

$$d_m(S_i, S_j) = \frac{1}{|matching_i|}\left|\{e \in matching_i | e \notin matching_j\}\right| \tag{4.9}$$

The matching dissimilarity metric is normalized with respect to the size of the set of the matching graph edges. This size is equal to the number of unique train units present at the shunting yard during the maintenance period.

The matching dissimilarity is a proper dissimilarity function. Due to the normalization, the range of the function is $(0, 1)$. The function evaluates to 0 only if the matching is the same in the two input schedules. More deviations in the matching lead to a larger value of the dissimilarity metric.

5

Rescheduling Algorithms

After having defined the schedule dissimilarity metrics, the focus will now be shifted to constructing reactive rescheduling policy based on those dissimilarity metrics. Using the knowledge of what makes a reschedule desirable, the goal is to create rescheduling algorithms which produce a feasible and desirable solution to the rescheduling problem. The dissimilarity function defined in chapter 4 can be used as a measure for the reschedule desirability, but an approach for creating reschedules that optimize for desirability still needs to be devised.

More formally, the main goal in this chapter is to devise a rescheduling algorithm f_{re}, as described in subsection 3.1.5. The rescheduling algorithm produces the reschedule S_{re}, which is feasible to the disturbed problem instance i_D. The disturbance model which is addressed by the rescheduling approaches in this chapter is the disturbance model in which a single train is delayed sufficiently long that the arrival order of the trains changes. This disturbance model is complex enough to force the use of repair methods different than right shift reschedule, as right shift reschedule is not flexible enough to deal with the described disturbance model. The choice of repair method is partial rescheduling. This method is flexible enough to deal with the described disturbance model. Furthermore, in contrast to methods such as full rescheduling, it also allows for dealing with disturbances occurring during the execution of the schedule. Finally, instance relaxation is disallowed in order to preserve schedule quality.

In this chapter, a total of four different approaches for finding feasible and desirable reschedules will be presented. The rescheduling policies will be based on the Local Search Simulated Annealing algorithm. The general Simulated Annealing algorithm will first be introduced in section 5.1. Then, the specific implementation of the Simulated Annealing scheduling approach for the TUSP, originally proposed by Van Den Broek[15], will be explained in section 5.2. Finally, the extensions to the Simulated Annealing approach for rescheduling purposes are proposed in section 5.3.

5.1. Simulated Annealing Local Search Approach

The reactive rescheduling strategy implemented in this paper is based on Local Search Simulated Annealing technique. This general local search meta-heuristic optimization technique is well suited for the rescheduling process, as local search techniques excel at exploiting the neighborhood space of solutions to the problems that they optimize. If a solution to the rescheduling problem can be found in close neighborhood of the pre-schedule, then the reschedule may display similarities with the pre-schedule.

Similarly to many other local search techniques, the general Simulated Annealing [17] algorithm works by improving upon some initial solution iteratively, in order to find a local optimum. The algorithm is

presented in Algorithm 1.

Algorithm 1 Simulated Annealing

```
1:  procedure SimulatedAnnealing(g : Objective function to minimize, T: Initial temperature)
2:      s ←s_0                                                    ▷ Current solution
3:      for all i ∈ {1..k} do
4:          s_candidate ←neighbor(s)                              ▷ Random neighbor
5:          if g(s_candidate) < f(s) then                         ▷ Immediately accept better solution
6:              s ←s_candidate
7:          else if exp((f(s)−f(s_candidate))/T) > rand(0,1) then  ▷ Stochastically accept worse solution
8:              s ←s_candidate
9:          end if
10:         T ←T · α                                              ▷ Decrease temperature
11:     end for
12:     return s
13: end procedure
```

The algorithm is initiated by computing an initial solution for the objective function g in line 2 of Algorithm 1. Then, a total of k iterations are performed in order to improve the solution, starting in line 3. In line 4, a new candidate solution is computed. The candidate solution is a *neighbor* of the solution s. Two solutions are neighbor if they are close to each other in the solution space, that is, there is a small change that can be applied to s in order to reach $s_{candidate}$. The concept of neighborhood in the context of TUSP is explained further in subsection 5.2.3. If the candidate solution is better than the previously found solution s, then it is accepted as the new solution, as seen in lines 5 and 6. If the candidate solution is not better, then it is either accepted or rejected stochastically, as seen in lines 7 and 8. In line 10, we see that the temperature T is gradually decreased by multiplying it with a constant α. The temperature decreases, since $0 < \alpha < 1$. The temperature decreases so that the probability to accept worse solution decreases over time.

5.2. Simulated Annealing for the TUSP

This section gives an introduction and overview of the Simulated Annealing approach for solving the TUSP by Van Den Broek [15]. This local search approach has been extensively used and researched at the NS, where it is usually called the Hybrid Integrated Approach (HIP). It is currently one of the best performing algorithms for solving the commonly occurring TUSP instances at the NS for selected shunting yards.

HIP is an implementation of the simulated annealing algorithm described in Algorithm 1. The implementation of HIP specifies the generation of initial solution, the objective function g and the neighborhood generation strategy, used for the selection of a random neighbor. The three implementation parts are described in detail in the following three subsections.

5.2.1. Initial Solution Generation

The initial solution is generated using various heuristics. First, the matching sub-problem is solved using a MIP solver for the Hopcroft-Karp matching algorithm [15]. Then, the services are assigned to service facilities using a simple ordering heuristic, the tasks are processed in order of increasing due time and are assigned to the earliest available service station [15]. The routing is then calculated using shortest path algorithm [15]. Finally, the parking sub-problem is solved by randomly assigning parking time during the train unit route [15].

The initial solution is usually of poor quality, as it is created using heuristics by solving the four TUSP sub-problems separately. The obtained initial solution may be impossible to execute in practice due to, for example, crossings of trains occurring in the schedule. The quality of the initial schedule is imporev

upon using the Simulated Annealing algorithm. According to Van Den Broek: ”One of the properties of simulated annealing is the tendency to move away quickly from the initial position in the search space, as it often accepts a deterioration in solution quality at the start of the search. As a result, any decent shunt plan success as initial solution.”[15].

5.2.2. Objective Function g

The objective function g in the HIP algorithm is calculated by computing the illegal and unwanted actions in a schedule and assigning a conflict cost or penalty cost to them. The conflict costs depict the costs which must be solved in order to make the schedule feasible, whereas the penalty costs represent unwanted actions that may get resolved but it is not necessary for the feasibility of the solution.

Both the penalty as well as the conflict costs are always calculated using a weight and the actual measure regarding the cost. The conflict cost measures include the following:

- Crossings - A movement which causes two train units to occupy the same space on a given track
- NonElectrifiedUsage - Usage of tracks which are not electrified by train units that require electrified tracks
- ArrivalDelay - A situation at the yard which prevents an incoming train to arrive on time
- DepartureDelay - Delay in the train departure
- TrackLengthViolations - A movement which causes a train to occupy a track which is too short
- CombineAtDeparture - A situation where two or more train units are combined at the track from which they depart

The conflict costs can be calculated using the cost measures and cost weights as follows:

$$
\begin{aligned}
\text{Conflict Costs} = \; & w_{Crossing} * |Crossings| \\
+ \; & w_{NonElectrifiedUsage} * |NonElectrifiedUsages| \\
+ \; & w_{ArrivalDelay} * |ArrivalDelays| \\
+ \; & w_{ArrivalDelayAverage} * AverageArrivalDelay \\
+ \; & w_{DepartureDelay} * |DepartureDelays| \\
+ \; & w_{DepartureDelayAverage} * AverageDepartureDelay \\
+ \; & w_{TrackViolations} * |TrackViolations| \\
+ \; & w_{TrackLengthViolation} * |TrackLengthViolations| \\
+ \; & w_{TrackViolationDuration} * TrackViolationDuration \\
+ \; & w_{CombineAtDeparture} * |CombineAtDeparture|
\end{aligned}
$$

The penalty costs consist of the following:

- EmptyShuntMoves - A shunting move with a route length of 0
- IllegalParkingTime - A parking task scheduled at a track with parking restrictions

The penalty costs can be calculated as follows:

The objective function consist of both the conflict and penalty costs. The comparison of the costs of two schedules happens in lexicographical order, with the comparison of the conflict costs first. Thus, the total cost is a tuple consisting of the conflict and penalty costs.

$$\begin{aligned} \text{Penalty Costs} = \quad & w_{EmptyShuntMoves} \quad * \quad |EmptyShuntMoves| \\ + \quad & w_{IllegalParkingTime} \quad * \quad IllegalParkingTime \end{aligned}$$

$$TotalCosts = \{ConflictCosts, PenaltyCosts\} \tag{5.1}$$

5.2.3. Neighborhood Generation Strategy

The core of the Simulated Annealing algorithm is the random choice of neighboring solutions. The neighbors of a solution are calculated using one of the defined neighborhoods. A neighborhood is the collection of solutions that can be obtained from some solution s by applying a single, small operation to them. The neighborhoods can be categorized by the sub-problem that they operate on: matching, task scheduling, parking and routing. The routing sub-problem is quite unique here, as all the neighborhoods may change the routing of the trains, in contrary to the other sub-problems, which only acquire a change by their respective neighborhoods.

Notice that there are many neighborhoods defined, but in the following paragraphs, only the unique neighborhoods are summed up. There exist more neighborhoods, but they are a combination of two or more neighborhoods applied consecutively.

Matching Neighborhoods There is a single neighborhood for the matching sub-problem, namely the MatchingNeighborhood. This neighborhood consists of all solutions that are reachable from some solution s by exchanging the matching of two train units.

Task Scheduling The task scheduling neighborhood consists of four different neighborhoods:

1. ServiceMachineOrderNeighborhood - Swapping the order of two consecutive service tasks of a single resource
2. ServiceMachineSwapNeighborhood - Swapping the resource of two given service tasks
3. ServiceMachineSwitchNeighborhood - Switching the resource of a given service task
4. ServiceTrainOrderNeighborhood - Swapping the order of the service tasks of a given train unit

Parking The parking neighborhood consist of a total of four neighborhoods:

1. ParkingInsertAndReturnNeighborhood - Extends a parking task at location T_1 to a sequence of parking tasks at T_1, T_2 and then again T_1.
2. ParkingInsertNeighborhood - Insert a parking move
3. ParkingSwapNeighborhood - Swap parking location of two trains
4. ParkingSwitchNeighborhood - Switch the parking location of a single train to another location
5. MovementMergeNeighborhood - Remove a parking job
6. MovementSplitNeighborhood - Split a route into two and insert a parking between them

Routing Finally, the routing sub-problem is approached using two neighborhoods.

1. MovementShiftNeighborhood - Change the total order of movements
2. MovementToDepartureTimeSwitchNeighborhood - Change the time of the move to the departure position

5.3. Rescheduling Methods

The rescheduling process is similar to the scheduling process in general. In both scheduling as well as rescheduling process, the objective is to find a feasible schedule. During the rescheduling process, the schedule should also be desirable, by fulfilling similarity requirements. Furthermore, the rescheduling process creates a solution to the disturbed scenario, whereas the scheduling process creates a solution to the undisturbed scenario. In this chapter, a total of four different rescheduling methods for the rescheduling algorithm f_{re} will be devised.

There are multiple possible approaches for the rescheduling process. The rescheduling methods proposed in this paper are based on the Simulated Annealing described in section 5.1 and section 5.2. A total of four different methods are proposed and described. The partial rescheduling approach is presented in subsection 5.3.1 and the relative guided partial reschedule is presented in subsection 5.3.2. The two rescheduling approaches: schedule repair and guided schedule repair are presented in subsection 5.3.3 and subsection 5.3.4 respectively.

5.3.1. Partial Rescheduling

The first proposed rescheduling approach relies on using the existing scheduling algorithm in order to solve the disturbed problem instance. In the partial rescheduling approach, a new, disturbed problem instance i_D is generated from the initial problem instance i_i, the initial schedule s_{pre} and a given disturbance occurrence D.

In this approach, the disturbed problem instance i_D is calculated in two steps. Firstly, the state of the shunting yard is simulated up until t_D, the time of occurrence of the disturbance. Then, the disturbance is applied by changing the planned arrival time of the late train. The disturbed problem instance i_D is the outcome of those two steps. Notice that i_D is equivalent to the scenario that has to be solved at the shunting yard after an occurrence of a disturbance.

The instance i_D is computed by calculating the position of every train unit at t_D in the instance i_i with the schedule s_{pre}. If the train has already arrived before t_D in i_i and according to s_{pre} is located at track $track_k$, then in i_D, the train arrives at t_D at the location $track_k$. If the train is scheduled to arrive after t_D in i_i, then it is also scheduled to arrive at the same time in i_D. If a train unit is executing a movement during t_D in s_{pre}, then that movement is also present and finished in s_{re}.

Any service task that in i_i takes place before t_D is not present in i_D, but tasks that in i_i take place after t_D are still scheduled at the same time in i_D. If a service task is being executed at t_D in s_{pre}, then this service task execution is continued in s_{re}.

Due to the fact that i_D is also a valid TUSP instance, it can be solved using any TUSP solver, as long as the task and movement continuation can be guaranteed. In this case, the HIP solver is used. The local search simulated annealing approach is responsible for finding a feasible solution for the rescheduling problem.

Notice that the partial rescheduling approach satisfies the three feasibility requirements for a proper rescheduling algorithm as defined in subsection 1.3.2. The solution of the partial rescheduling algorithm is a solution to the disturbed initial problem instance. Furthermore, the simulation continues on from t_D, and so no actions are planned for $t < t_D$, nor are actions with $t < t_D$ changed.

Even though the partial rescheduling approach can be used to generate feasible reschedules, those reschedules may not be desirable at all. This is due to the fact that this approach does not consider schedule similarity at all. Therefore, the generated reschedules may not be similar to the pre-schedule and thus be undesirable.

5.3.2. Guided Partial Rescheduling

The second rescheduling approach, guided partial rescheduling, is an extension to the partial rescheduling approach described in subsection 5.3.1. The partial reschedule approach does not consider schedule similarity, and thus also schedule desirability, at all. Both g_{pro} as well as g_{re} are missing.

The proposed extension is largely based on the dissimilarity measures defined in section 4.3. The idea is to extend the objective function g_{re} with the dissimilarity measures. In the partial rescheduling approach, the simulated annealing algorithm is responsible for creating a feasible schedule. In order to create the reschedules not only feasible but also desirable, the algorithm is guided during its execution to focus more on similar schedules. In order to achieve this, the dissimilarity costs are first defined in Equation 5.2.

$$\begin{aligned} \text{Dissimilarity Costs} = \; & w_{TaskStartingTimeDissimilarity} * d_t \\ + \; & w_{TaskLocationDissimilarity} * d_l \\ + \; & w_{RoutingDissimilarity} * d_r \\ + \; & w_{MatchingDissimilarity} * d_m \end{aligned} \tag{5.2}$$

The dissimilarity functions are summed together using weights, similarly to conflict and penalty costs. The dissimilarity is calculated between the pre-schedule and the candidate schedule. The sum of the conflict and penalty costs is minimized during every iteration of the algorithm. Therefore, the algorithm will be guided to search for both feasible as well as similar, desirable solutions.

While there are many different ways to include the dissimilarity costs into the objective function of the HIP algorithm, in this approach it has been decided to add the dissimilarity costs to the conflict costs of the original objective function of the simulated annealing algorithm. Thus, the following holds.

$$TotalCosts = \{ConflictCosts + DissimilarityCosts, PenaltyCosts\} \tag{5.3}$$

The advantage of summing the conflict costs with dissimilarity costs is the fact that the dissimilarity costs are optimized for during every iteration step of the algorithm. The disadvantage is that the conflict costs are worth relatively less, since both the conflict costs and dissimilarity costs are being optimized for. In short, the guided partial reschedule approach optimizes more for schedule desirability as opposed to schedule feasibility, when compared to the partial reschedule approach.

Guided partial reschedule is inherently closely related to the partial reschedule approach. Similarly to partial rescheduling approach, the guided partial rescheduling approach satisfies the three feasibility requirements of a rescheduling algorithm subsection 1.3.2. Unlike partial rescheduling approach, the guided partial rescheduling approach also attempts to satisfy the desirability requirement by the extension of g_{re}.

5.3.3. Schedule Repair

The previous two approaches have focused on both schedule feasibility and desirability. In the schedule repair approach, the main focus will, again, be the schedule desirability. Whereas the guided partial reschedule subsection 5.3.2 achieved similarity by modifying the objective function of the simulated annealing algorithm, the schedule repair approach will achieve the same property by modifying the initial solution generation step of the simulated annealing algorithm.

The initial solution of the HIP algorithm is created using various heuristics, as described in subsection 5.2.1. The quality of the initial solution is generally poor, and the simulated annealing is used to create a better quality solution. In the case of rescheduling, a solution with reasonable quality is already present, namely the pre-schedule s_{pre}. The pre-schedule can be used as the initial solution for a disturbed scenario. Then, simulated annealing can be used to solve any conflicts that result from the

fact that the pre-schedule is a solution to the undisturbed scenario and not the disturbed one.

The schedule repair approach is not an extension of either the guided partial reschedule or the partial reschedule approach. Special care must be taken to ensure that the approach satisfies reschedule feasibility requirements as described in subsection 1.3.2. This is done in two steps: firstly by modifying the neighborhood search algorithm of HIP and secondly by modifying the problem instance and the initial solution.

The random neighborhood search of HIP generates a neighboring solution from a source solution by modifying an action of the source solution. In general, the neighborhood search may modify any action in the source schedule. This should not be allowed when dealing with rescheduling, and so, two modifications are made. Firstly, no action with starting time $t < t_D$ may be changed by the neighborhood search. Secondly, the actions created or modified by the neighborhood search may not be placed at $t < t_D$.

The second step is the modification of the problem instance and the initial solution to reflect the disturbed scenario. The disturbance is applied to the problem instance by delaying the train arrival. This causes a problem, since updating just the problem instance but not the initial solution causes the initial solution to be infeasible. Usually, such infeasibility would be solved using the simulated annealing algorithm. This is not possible in the case, due to the nature of the infeasibility caused by the disturbance. The infeasibility is caused by the fact that two trains (one of which is the delayed train) are scheduled to arrive in some specific order, but in the initial solution schedule, they arrive in the opposite order. This is a situation which does not occur when the solution is generated using heuristics nor does it occur after applying changes by neighborhood search. The neighborhoods of HIP do not include one which would be able to resolve the aforementioned issue. Therefore, the issue is addressed before the algorithm is used. Namely, the arrival order of the late train and the train that arrives after the late train in the pre-schedule is reversed in the initial solution schedule.

The two steps of modifying the HIP neighbor search and modification of the problem instance and the initial solution cause the schedule repair to become a proper scheduling approach, due to the fact that it satisfies the rescheduling algorithm requirements as described by subsection 1.3.2.

The schedule repair approach may be suitable to find not only feasible solutions, but also desirable solutions. This is due to the fact that the algorithm begins its search with a very desirable solution, namely the pre-schedule. During the search, the desirability of the solution may degrade, since no notion of desirability exists during the run-time of the schedule repair approach.

5.3.4. Guided Schedule Repair

The schedule repair approach begins its search with a very desirable solution, namely the pre-schedule. During the run-time of the simulated annealing algorithm, the desirability of the solution may deteriorate. To address this issue, the notion of stability can be introduced to the run-time of the algorithm.

The proposed guided schedule repair approach is an extension to the schedule repair approach. Similarly to the guided partial rescheduling approach, the guided schedule repair approach relies on guiding the simulated annealing algorithm during its runtime to focus on schedules similar to the pre-schedule. The proposed method to achieve this is to extend the objective function of the HIP algorithm to include the dissimilarity costs. The dissimilarity cost is defined the same way as it was for guided partial rescheduling, namely:

$$\begin{aligned} \text{Dissimilarity Costs} = \; & w_{TaskStartingTimeDissimilarity} * d_t \\ + \; & w_{TaskLocationDissimilarity} * d_l \\ + \; & w_{RoutingDissimilarity} * d_r \\ + \; & w_{MatchingDissimilarity} * d_m \end{aligned} \tag{5.4}$$

The objective function is extended to the following form:

$$TotalCosts = \{ConflictCosts + DissimilarityCosts, PenaltyCosts\} \tag{5.5}$$

Similarly to the guided partial rescheduling method, this approach is chosen such that the algorithm optimizes for schedule desirability, possibly at the cost of schedule feasibility.

Since the guided schedule repair is an extension of the schedule repair approach and so it also satisfies the requirements of a proper rescheduling algorithm as defined by subsection 1.3.2. The guided schedule repair optimizes for schedule feasibility by starting with a desirable initial solution as well as minimizing dissimilarity during the run-time of the algorithm.

The four proposed rescheduling algorithms differ mostly in how the desirability of the schedule is achieved. The four algorithms are examples of the rescheduling function f_{re} of the abstract rescheduling framework. The differences in achieving schedule desirability is achieved through different definitions of g_{re} as well as the implementation of the initial solution of the simulated annealing algorithm. The approaches are summarized with that property in mind in Table 5.1.

Table 5.1: Summary of the Rescheduling Approaches

	Partial Rescheduling	**Guided Partial Rescheduling**	**Schedule Repair**	**Guided Schedule Repair**
Desirability optimization during run-time	-	Yes	-	Yes
Desirable initial solution	-	-	Yes	Yes

In short, four different rescheduling algorithms have been proposed. The performance of the algorithms is not yet known. The four algorithms represent four different approaches possible for achieving schedule desirability. It is yet to be determined whether any of the proposed algorithms performs well enough to create desirable schedules. It is also not known whether the complex approaches for obtaining reschedule desirability are necessary at all, or if the partial rescheduling algorithm is able to produce desirable reschedules. Therefore, the performance of the proposed algorithms will be further investigated. In chapter 6, the experimental setup for the performance analysis will be explained, and in chapter 7, the results will be presented.

6

Experimental Setup

In chapter 5 a total of four rescheduling approaches have been devised. Furthermore, the objective function to evaluate the desirability of a reschedule has been proposed in chapter 4. The rescheduling performance of the proposed solutions can and should be measured. The performance of the rescheduling algorithm is multi-objective, as a proper reschedule should be both feasible as well as desirable. The reschedule is feasible if no illegal actions are planned in the reschedule, or in other words, if the conflict costs defined in subsection 5.2.2 of the reschedule are 0. A reschedule is desirable if it is similar to the pre-schedule, following the definitions in chapter 4.

In this chapter, the experimental setup for measuring the rescheduling algorithm performance will be proposed. In order to perform the rescheduling experiments, various components are needed. Following the definitions of the proactive-reactive scheduling framework from chapter 3, the following components need to be defined:

- Problem Instance Model I
- Disturbance Model D
- Objective Function g_{re}
- Pre-schedule Generation Algorithm f_{pro}
- Rescheduling Algorithm f_{re}

The implementation of the five components will be explained in order. First, the problem instance model will be proposed in section 6.1. Then, the implementation of the disturbance model will be explained in section 6.2. Thirdly, the objective function g_{re} used to evaluate the rescheduling performance will be explained in section 6.3. Fourthly, the pre-schedule generation will be presented in section 6.4. Finally, the rescheduling algorithm experiment will be presented in section 6.5.

6.1. Problem Instance Model I

The problem instances are generated using the tool called the Instance Generator, developed by NS. The tool is able to generate problem instances that closely resemble real problem instances. A proper schedule is generated in which trains arrive at a certain time. The trains require service tasks to be executed on them before they depart at the time of departure. The instances generated by the tool can be considered to be realistic, according to the experts at NS.

The problem instances are generated for the Kleine Binckhorst shunting yard. This shunting yard is the one that is most commonly used for testing of scheduling and rescheduling algorithms for the TUSP at NS. The nominal capacity of this yard is 16 train units. This is the capacity for which the HIP algorithm can solve over 95% of instances generated by the instance generator. The percentage of instances

that can be solved using HIP decreases with the number of train units. In other words, a larger number of train units cause the problem instances to become more difficult. At 18 train units, around 50% of problem instances can be solved, and at 19 train units, less than 10% of the problem instances are solvable.

Ideally, the rescheduling performance should be tested on instances with the number of train units varying from 0 to 20 (at 20 train units almost no instances can be solved). Due to a large run-time of the algorithm, this is not feasible. Therefore, a choice is made to include two different values of train units. The decision to measure the rescheduling performance on instances with 16 and 18 train units has been made. The choice of 16 train units is made due to 16 being the nominal capacity of the shunting yard and 18 is the largest number of train units where a large number of instances is still solvable.

A total of 25 problem instances with 16 train units are generated. Similarly, a total of 25 problem instances with 18 train units are generated as well. All the measurements of rescheduling performance are calculated on these 50 problem instances.

6.2. Disturbance Model D

The disturbance model assumed in this paper is based on a single late train. The train is delayed so much that the arrival order of trains is changed. The train is delayed just so that the order changes, no more delay is implemented than needed. The implementation of the disturbance model is described in Algorithm 2.

Algorithm 2 Train Order Disturbance Generation

1: **procedure** GenerateTrainOrderDisturbance(T_i : Arriving train, T_j : First train arriving after T_i)
2: $a_i \leftarrow$ arrival time of T_i
3: $a_j \leftarrow$ arrival time of T_j
4: **return** $Delay(T_i, a_j - a_i + c)$ ▷ Delay train T_i to arrive c minutes after T_j
5: **end procedure**

In order to generate disturbances according to the disturbance model, for every arriving train T_i except the last one, a single disturbance is created. In this disturbance, the train T_i arrives c minutes after train T_j, where T_j is the first train that arrived after T_i in the original problem instance.

An important exception to the rule described above is the case where there is no time between the arrival of T_j and T_{j+1} for the arrival of T_i. This occurs when the time difference between the arrivals of T_j and T_{j+1} is less than 5 minutes. In such cases, scheduling T_i to arrive between T_j and T_{j+1} is meaningless, since such situations are not solvable by HIP, as they do not occur in real life scenarios. In such cases, T_i is delayed to arrive after T_{j+1}.

Note that a disturbance is generated for each arriving train in the schedule, with the exception of the last one. Therefore, a total of around 12 disturbances is generated for each problem instance with 16 train units. This is due to the fact that a disturbance is generated for each incoming train and not a train unit. A total of 16 disturbances is generated for problem instances with 18 train units.

6.3. Objective Function g_{re}

The objective function for the rescheduling algorithm is multi-objective. A proper reschedule must be both feasible and desirable. To achieve this, the total costs as defined in subsection 5.2.2 are used as the objective function, so the following holds:

$$TotalCosts = \{ConflictCosts + DissimilarityCosts, PenaltyCosts\} \tag{6.1}$$

Once again, the reschedule is feasible only if the conflict costs are 0 and it is desirable if the dissim-

ilarity costs are low. The penalty costs are used in lexicographical manner, in order to solve ties for reschedules with the same value for the sum of conflict and dissimilarity costs. Notice that to calculate the conflict, dissimilarity and penalty costs, many weights regarding the severity of violations need to be specified. For the conflict and penalty costs, the weights specified by the experts at NS for the HIP solver are used. The conflict weights are presented in Table 6.1 and the penalty weights are presented in Table 6.2.

Table 6.1: Conflict Weights

$w_{Crossing}$	=	10
$w_{NonElectrifiedUsage}$	=	10
$w_{ArrivalDelay}$	=	15
$w_{ArrivalDelayAverage}$	=	0.002
$w_{DepartureDelay}$	=	25
$w_{DepartureDelayAverage}$	=	0.005
$w_{TrackViolations}$	=	12
$w_{TrackLengthViolation}$	=	0.01
$w_{TrackViolationDuration}$	=	0.0002
$w_{CombineAtDeparture}$	=	15

Table 6.2: Penalty Weights

$w_{EmptyShuntMoves}$	=	0.02
$w_{IllegalParkingTime}$	=	0.01

The dissimilarity weights are introduced in this paper and thus it is not possible to rely on previous work by the NS experts to determine the weights for those. The main focus is put on ensuring the temporal similarity, so the weight for the service time dissimilarity is chosen to be larger than the other weights, namely 10. The weights for the other three dissimilarity measures have been chosen to be 1. The weights have been chosen experimentally, no formal parameter selection strategy has been used. The weights are generally quite low compared to the conflict weights. The expectation is that the algorithm will focus foremost on the feasibility of the solution first. The values are again presented in the Table 6.3.

Table 6.3: Dissimilarity Weights

w_{d_t}	=	10
w_{d_l}	=	1
w_{d_r}	=	1
w_{d_m}	=	1

6.4. Pre-Schedule Generation Algorithm f_{pre}

The next component needed for the reschedule experiments is the pre-schedule generation algorithm. The pre-schedule generation algorithm is HIP, and thus, the pre-schedules are generated from problem instances using the HIP solver. The solver is set up to solve for feasibility with a time limit of 10 minutes. If a unique solution is found within this time limit, then this solution is saved as a pre-schedule for the given scenario. On the other hand, if a solution is not found or a duplicate solution is found, then the pre-schedule is discarded.

For each of the 50 scenarios, a total of 20 unique pre-schedules are generated. The solver might be used on a scenario more than 20 times. This is done due to the fact that the solver is probabilistic and

so it might fail to find a pre-schedule or find a pre-schedule multiple times. In the case that after 100 runs no 20 unique pre-schedules are found, the scenario is discarded. The procedure is also presented in Algorithm 3.

Algorithm 3 Pre-schedule Generation

```
1:  procedure GeneratePreSchedules(i : Problem Instance n : Integer)
2:      PreSchedules ← ∅                                  ▷ The generated pre-schedules
3:      i ← 0
4:      while i < n + m and |PreSchedules| < n do
5:          S ← HIP(i)                                    ▷ HIP is the SA solver
6:          if S is feasible then
7:              PreSchedules ← PreSchedules ∪ S
8:          end if
9:          i ← i + 1
10:     end while
11:     if |PreSchedules| < n then
12:         reject s
13:     end if
14:     return PreSchedules
15: end procedure
```

6.5. Rescheduling Algorithm f_{re}

The four rescheduling algorithms devised in section 5.3 will be tested. The four algorithms will be tested on all the generated problem instances, disturbances and pre-schedules. The general run-time of the test is presented in Algorithm 4.

Algorithm 4 Rescheduling Algorithm Run-time

```
1:  procedure Solve(f_re: Rescheduling Algorithm)
2:      for each I do                                     ▷ Problem Instances
3:          for each S_pre of I do                        ▷ Pre-schedules
4:              for each d of I do                        ▷ Disturbances
5:                  I_D ← disturb(I, d)                   ▷ Disturbed Instance
6:                  for i ∈ {1..20} do
7:                      S ← f_re(I_d, S_pre)
8:                  end for
9:              end for
10:         end for
11:     end for
12: end procedure
```

It can be seen that each of the four algorithms is ran multiple times on the various problem instances, pre-schedules and disturbances. This can be seen in lines 2 through 4. Then, on line 5, the disturbed problem instance is calculated. Each disturbed instance is solved using the rescheduling algorithm 20 times, as seen in lines 6 and 7. This is done due to the fact that the rescheduling algorithms proposed earlier are all probabilistic and so they might fail to find a solution, or find a solution of poor quality if just a single run is executed. For this reason, the calculation is done 20 times.

Notice that due to the quadruple for loop construction in Algorithm 4, every algorithm is ran more than 300 000 times. Every iteration is allowed to run for 30 seconds. This causes a very long run-time. This is addressed by performing the computations in parallel. The experiments have been ran in parallel, 24 at a given time, using a single computer with a 24-thread processor.

7

Results

In chapter 6, the experimental set-up for the performance analysis of the four rescheduling algorithms defined in chapter 5 has been described. The four rescheduling algorithms approach the achievement of schedule desirability differently. The performance analysis is executed in order to determine whether any of the four rescheduling algorithms is adequate to produce desirable reschedules and whether the proposed extensions indeed increase the desirability of the found solutions.

A large amount of data can be produced using the experimental setup used described in chapter 6. The four proposed rescheduling algorithms have been ran over 300 000 times each. In this chapter, the results of those experiments are presented in a concise way.

Firstly, the performance of the proposed rescheduling strategies with regards to schedule feasibility is presented in section 7.1. Then, the performance with regards to schedule desirability is presented in section 7.2. Finally, the performance with respect to the service task lateness is further investigated in section 7.3.

7.1. Feasibility

The first criterium to look for at the experiments is the feasibility of the solutions. The goal is to determine whether the rescheduling algorithms are producing feasible reschedules. If a reschedule is not feasible, then regardless of whether it is desirable, it can not be implemented in practice.

Due to the fact that each of the unique experiment set-ups is repeated 20 times, a total of two different measures for feasibility are proposed. In the first measure, the feasibility is measured for all the experimental samples. This includes the 20 repetitions for each experiments. The feasibility is thus calculated by dividing the number of feasible solutions found by the total number of solutions attempted. On the other hand, the feasibility score can also be aggregated, due to the 20 repetitions for each experiments. This is calculated by first calculating the feasibility of each experiment without the 20 repetition redundancy. Such experiment is assumed to be feasible if at least one of the 20 repetitions produces a feasible solution. This approach is reasonable, due to the fact that in practice, the algorithm may be ran multiple times as well. The results are presented in Table 7.1.

Table 7.1: Performance of the Rescheduling Algorithms: Feasibility

		Partial Reschedule	Guided Partial Reschedule	Schedule Repair	Guided Schedule Repair
Feasibility	**16 TU**	74.4%	66.6%	94.5%	93.9%
	18 TU	32.4%	29.5%	77.6%	77.5%
Aggregated Feasibility	**16 TU**	100%	100%	100%	100%
	18 TU	56.7%	48.3%	96.4%	96.3%

7.1.1. Discussion

The performance of the rescheduling algorithms with regard to schedule feasibility show that in general, all four of the algorithms can be used to find feasible reschedules. It can be noted that the schedule repair and the guided schedule repair algorithms outperform or perform equally as the partial reschedule and guided partial reschedule algorithms, in all the feasibility categories for all the train unit configurations. The use of the pre-schedule as the initial solution greatly increases the performance of the rescheduling algorithms with regards to feasibility.

On the other hand, guiding the algorithms by integrating dissimilarity costs into their objective function does not increase the algorithm performance with regards to feasibility. This change has very little effect between the schedule repair and guided schedule repair algorithms, but it decreases the performance of the guided partial reschedule as compared to partial reschedule.

As expected, the difficulty of computing the reschedules for problem instances with 18 train units is more difficult than it is with 16 train units. The feasibility performance for 16 train units for the schedule repair and guided schedule repair methods is unexpectedly high. More than 93% of reschedules generated by either of the two methods in 30 seconds are feasible. Without any modifications, around 80% of schedules generated by HIP are feasible. Thus, the rescheduling performance is better than the scheduling performance. This could be due to the fact that starting with a high quality initial solution aids the process a lot, but this can not be concluded with certainty. This is because a rescheduling process in general is not equivalent to a scheduling process, as the rescheduling process only schedules a subset of tasks that a scheduling process needs to schedule.

Finally, it can be seen that all the algorithms can be used to generate reschedules in the case of 16 train units, when the rescheduling process is ran 20 times. In such case, all the algorithms produce at least one feasible solution to the rescheduling problem. This changed in the case of 18 train units. In such case, the partial reschedule and guided partial reschedule methods produce a feasible reschedule within 20 iterations in just 56.7% respectively 48.3% of the cases. This is considerably worse than the schedule repair and guided schedule repair methods.

7.2. Desirability

The second performance characteristic of the rescheduling algorithms is the desirability of the output schedule. All the metrics presented in this section are aggregated with respect to the redundancy of 20 repetitions of each of the experiments. From the 20 experiment repetitions, only the reschedule with the lowest dissimilarity costs (as defined in section 6.3) is used for the metric measurement. Notice that a reschedule may only be selected if it is also feasible, unfeasible solutions are discarded. If an experiment has no feasible solutions, it is discarded and not counted towards the statistics.

The desirability of the schedules is measured in similarity terms defined in section 4.3. The sum of task latenesses is measured in hours. In the ideal rescheduling solution, this sum is 0. Larger values of this metric decrease the desirability of a reschedule. The other three metrics: task location deviation, routing and matching are presented as percentages. Values close to 0% are a characteristic of desirable reschedules, whereas values close to 100% characterize undesirable reschedules.

Table 7.2: Performance of the Rescheduling Algorithms: Desirability

		Partial Reschedule	**Guided Partial Reschedule**	**Schedule Repair**	**Guided Schedule Repair**
Sum of Task Lateness	**16 TU**	8:32 hours	2:09 hours	3:05 hours	1:42 hours
	18 TU	9:47 hours	2:32 hours	3:31 hours	1:48 hours
Task Location Deviation	**16 TU**	58.2%	30.2%	2.39%	2.44%
	18 TU	60.2%	29.5%	2.73%	2.30%
Routing	**16 TU**	95.3%	83.1%	4.19%	6.98%
	18 TU	94.6%	85.2%	5.67%	7.17%
Matching	**16 TU**	38.3%	17.1%	1.37%	0.46%
	18 TU	49.2%	21.9%	3.84%	0.65%

7.2.1. Discussion

The performance of the rescheduling algorithms from the desirability perspective shows that the partial reschedule as well as guided partial reschedule methods produce reschedules that are insufficiently similar to be considered as desirable. In both the cases of 16 as well as 18 train units, these two rescheduling algorithms produce reschedules that are not similar to the respective pre-schedules with regards to task location deviation, routing and matching. In the best case, at least 29.5% of the service tasks are relocated, 83.1% of the route is changed and the matching of 17.1% train units is changed. These performance metrics are worse for the partial reschedule and guided partial reschedule methods than they are for the schedule repair and guided schedule repair methods.

The schedule repair and guided schedule repair methods perform significantly better with respect to the task location deviation, routing and matching. The performance metrics show that the reschedules produced with the rescheduling methods based on schedule repair perform better than their respective counterparts based on partial rescheduling. It can not be concluded without further experimentation whether the reschedules produced by these two methods are indeed regarded as desirable. It can, however, be concluded that the choice of the initial solution greatly increases the rescheduling performance with regards to desirability.

The sum of task latenesses metric shows a different performance than the other three metrics. It is the only metric where the guided partial reschedule outperforms the schedule repair. The guidance of the local search algorithm towards more similar solutions aids the performance of the algorithms with

regard to the sum of task lateness metric.

Another difference of the sum of task lateness metric is the fact that it is not a percentage and so it is not trivial to determine whether the results are any close to the lower bound or not. In the case of the three other metrics, the lower bound is 0%. It is not the case that the sum of task lateness has a lower bound of 0. In order to determine whether the sum of task lateness of the guided schedule repair method of 1:42 hours for 16 train units or 1:48 hours for 18 train units is close to the lower bound, more investigation is performed in section 7.3.

7.3. Sum of Task Lateness Performance

The rescheduling algorithm may change the starting time of any service task that occurs after the rescheduling time. Ideally, none of the service tasks should be incur lateness, however, it can be concluded from the results in section 7.2 that services do incur lateness after rescheduling. The results of the task lateness performance are not trivial to analyze, as it is not clear what the theoretical best performance is. In order to determine whether the results obtained in section 7.2 are close to theoretical optimum, the lower bound for task lateness will be derived in this section. The analysis in this section is based on the guided schedule repair method using both 16 as well as 18 train units.

7.3.1. Source of the Delay

In order to correctly analyze the performance with respect to the sum of task lateness, the source of the task lateness will first be analyzed. The sources of the delay can be categorized by the train unit they belong to. This can be either a train unit that arrived late, or a train unit that did not arrive late. This distinction is useful for further analysis. Most importantly, the number of services of the delayed train unit is much smaller than the number of services of non-delayed train units.

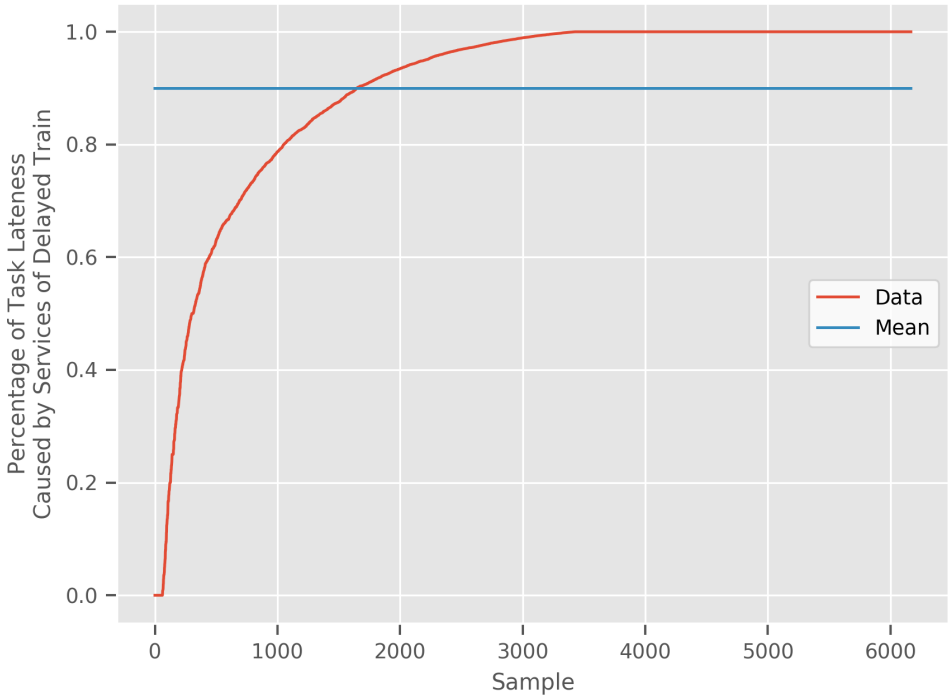

Figure 7.1: Sources of Task Lateness

The composition is calculated by computing the ratio of the sum of the lateness of the services of the delayed train and the sum of total lateness. The result is displayed in Figure 7.1. On average,

the services of the delayed train contribute for 91.2% of the total task lateness. Furthermore, the plot shows that for most of the samples, the task lateness of the services of the delayed train contribute for a large part of the total task lateness. Therefore, the task lateness of the services of the delayed train unit approximates the total task lateness of the reschedule scenario. For this reason, an adjustment to the prediction model is proposed. Instead of attempting to predict the total task lateness of a given reschedule scenario, the task lateness of the services of the late train can be predicted.

It can be concluded that most of the task lateness, in general, is caused by the services of the delayed train. For this reason, further investigation will be performed into why the services of delayed trains are being delayed.

7.3.2. Lower Bound for the Task Lateness of a Delayed Train

It has been concluded that most of the task lateness of most of the instances are caused by the services of the delayed train. In this subsection, the reason for those task latenesses will be investigated. This will be done hand in hand with deriving a lower bound for the task lateness of the delayed train.

The lower bound of a task lateness will be derived using Figure 7.2. In this figure, two different pre-schedules are displayed. In the first pre-schedule, the slack s_1 between the arrival and the inspection task is much smaller than the arrival delay due to a disturbance D. Since the inspection task must be performed after the arrival, the optimal reschedule has a task lateness for the inspection task of at least $D - s_1$.

On the other hand, in pre-schedule 2, the slack s_2 between the arrival and the inspection service task is equal to the arrival delay due to the disturbance. The optimal reschedule for pre-schedule 2 incurs no task lateness for the inspection service task, since the slack s_2 is sufficiently large to negate the effects of the arrival delay D.

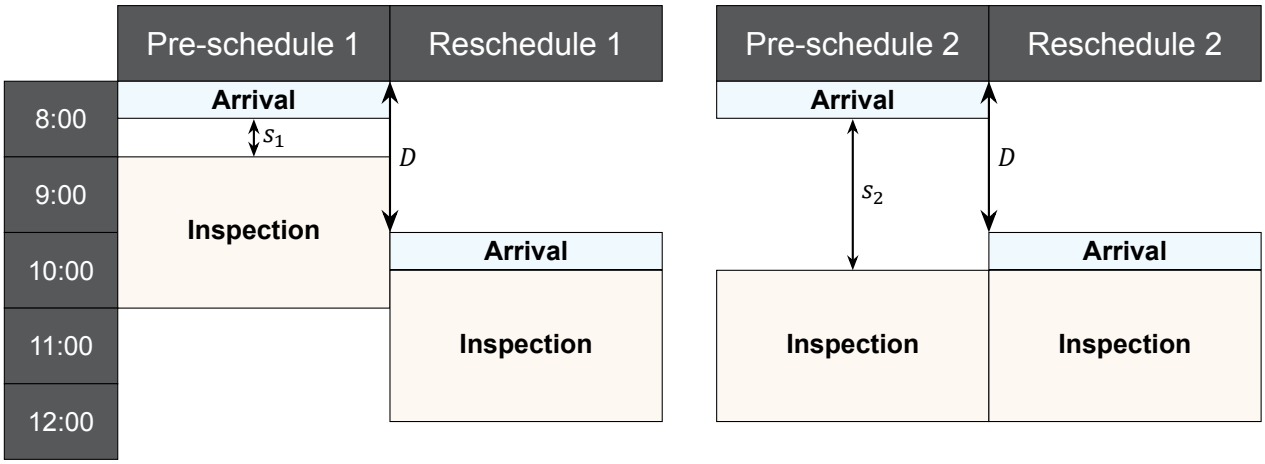

Figure 7.2: Task Delay Lower Bound

This example can be used to derive a lower bound for a task lateness. The minimal task lateness for a service task of a delayed train is defined in Equation 7.1.

$$\text{lateness lower bound} = \begin{cases} D - s & \text{if } D - s \geq 0 \\ 0 & \text{otherwise} \end{cases}. \tag{7.1}$$

Notice that this lower bound is not as tight in the case where the late train has more than one service task. This is displayed in Figure 7.3.

In this figure, it can be seen that the the cleaning task has a slack s_1 to the arrival time and $s_1 > D$. According to the lower bound equation in Equation 7.1, the lower bound for the cleaning task lateness should be 0. This is, however, not the case, as the cleaning task incurs a delay due to the fact that the

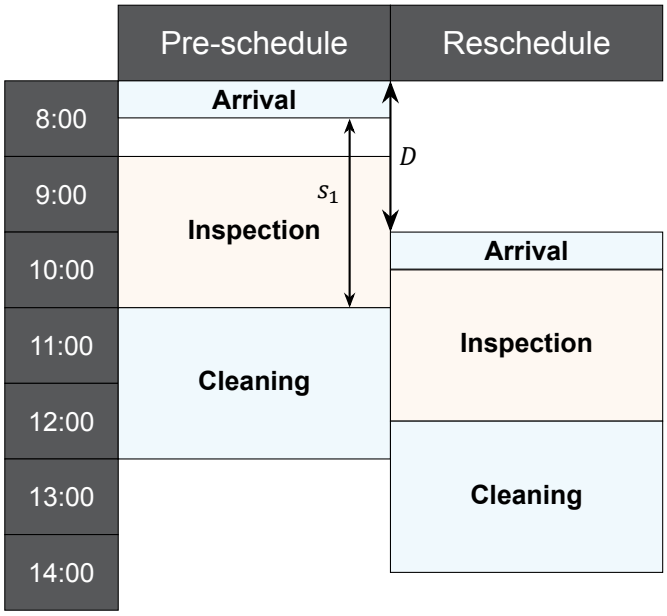

Figure 7.3: Task Delay Lower Bound with Two Service Tasks

inspection task is delayed. This shows that the lower bound is not as tight for the second task of a train unit as it is for its first task. Notice that in none of the problem instances does a train unit occur with more than 2 tasks.

Furthermore, notice that the lower bound does not take into account any pauses or time needed for routing between the arrivals and the tasks. It also does not take into account possible situations where other tasks occupy all possible resources.

Using the knowledge of the service task lower bound, it is now possible to investigate the results obtained in section 7.2 more closely.

7.3.3. Desirability of the Reschedules with respect to Task Lateness

The lower bound for the task lateness has been derived and it has been concluded that most of the task lateness occurs due to services of the delayed train. This knowledge can be used to determine whether the performance of the rescheduling algorithms with respect to task lateness is reasonable.

Firstly, it is considered what fraction of the total lateness can be explained by the service task lateness lower bound. This analysis is performed only on the results from the guided schedule repair rescheduling algorithm. On average, more than 60% of the total lateness occurs due to the effects of the lower bound. This is displayed in Figure 7.4.

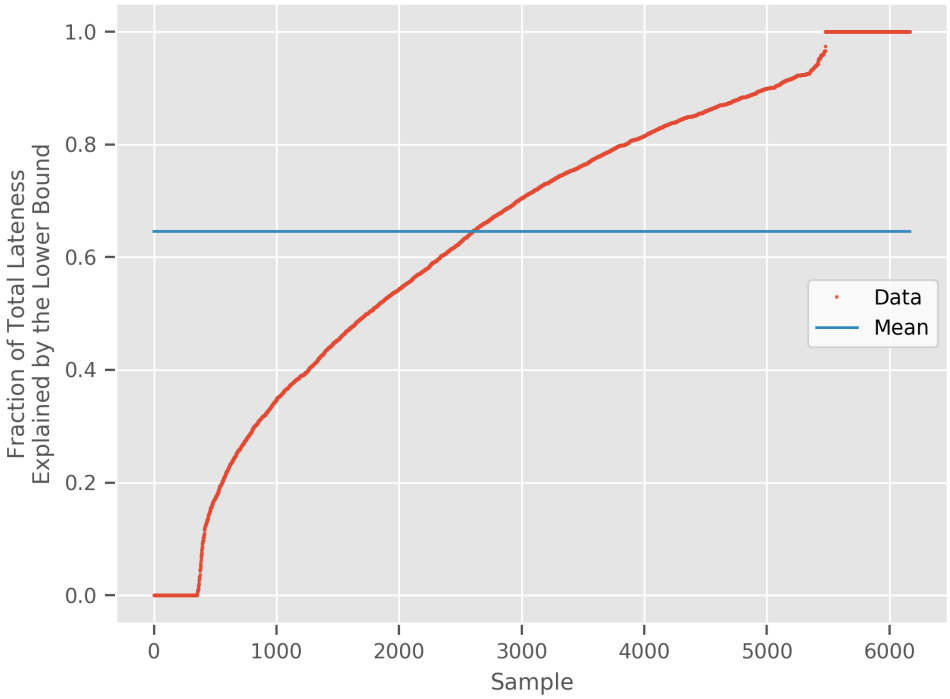

Figure 7.4: Total lateness explained by the lower bound

Since a large percentage of the total lateness can be explained by the theoretical lower bound, it can be concluded that the guided schedule repair performs well with respect to the sum of task lateness performance metric. The results of 1:42 hours task lateness for experiments with 16 train units are thus largely explained by the quite loose theoretical lower bound.

Lastly, a measurement is performed to determine whether larger values of the arrival to service task slack indeed cause the lateness of tasks to decrease. In order to do this, a correlation study is performed. Due to the fact that the theoretical lower bound is looser for the second task of a train unit than it is for its first service task, the correlation study is performed separately on the first service tasks and second service tasks. The correlation is measured between the slack of a service task and the actual lateness incurred in the reschedule. Note that for this correlation study, normalized values for the slack and the lateness have been used. The results can be found in Figure 7.5

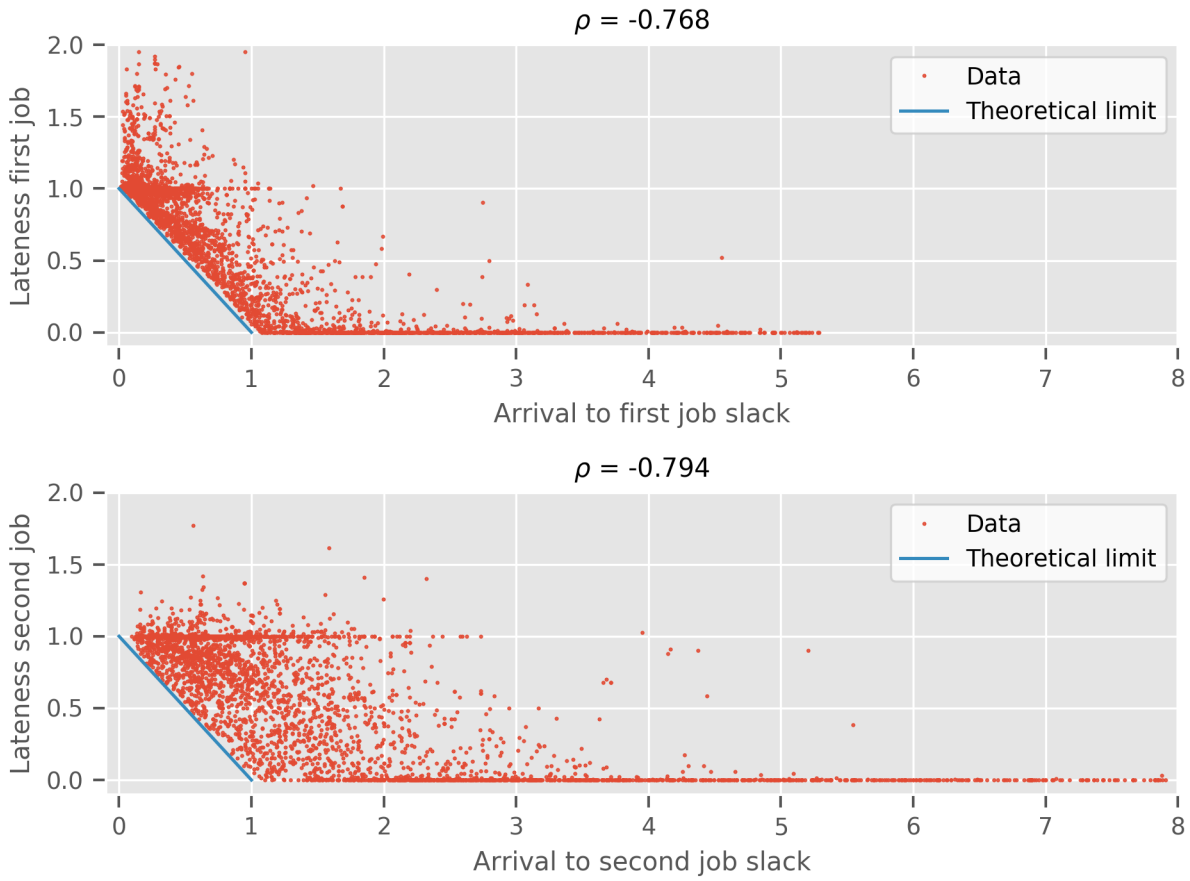

Figure 7.5: Correlation between slack and the lateness incurred due to rescheduling

The two measures show a strong correlation with Spearman's rank correlation coefficient of $\rho = -0.768$ in the case of the first service task of a train unit and $\rho = -0.794$ in the case of the second service task of a train unit. Thus, we can conclude that in order to reduce the total lateness incurred due to rescheduling, a viable strategy would be to increase the slack between the arrival and the starting time of the service tasks.

This is a powerful finding, since it can be used to create pre-schedules that are more flexible with regards to the total lateness incurred due to rescheduling.

8

Conclusion

The train maintenance of the train fleet of the Nederlandse Spoorwegen occurs at certain shunting yards. The maintenance schedule is created beforehand in order to guarantee the feasibility of the maintenance plan. The execution of the maintenance schedule is not an entirely deterministic process and the occurrence of certain disturbances can cause the maintenance schedule to become infeasible. In this research, a reactive method do deal with unforeseen disturbances has been devised. This method is based on simulated annealing local search approach. This method is a member of the broader category of proactive-reactive scheduling approaches, which has also been defined and described. Finally, the properties of the pre-schedules that reduce temporal flexibility have been analyzed.

8.1. Research Questions

Can proactive-reactive rescheduling method be used to deal with disturbed TUSP schedules?

In this research, a total of four reactive rescheduling strategies based on simulated annealing local search algorithm have been proposed. The guided schedule repair method has displayed superior performance from the four proposed rescheduling strategies. It has been demonstrated that using this strategy, it is possible to deal with disturbances of varying severity. The resulting reschedules are feasible and similar to the pre-schedule. Furthermore, it has been shown that the slack between the train unit arrival and the starting time of its service task is negatively correlated with the temporal flexibility of the rescheduling process. This insight can be used to devise proactive rescheduling methods.

What are the components of a proactive-reactive scheduling algorithm and how are they relevant for the TUSP?

In general, the proactive-reactive scheduling strategy has been demonstrated to contain five components:

1. Problem Instance Model I
2. Disturbance Model D
3. Objective Functions g_{pro} and g_{re}
4. Pre-schedule generation algorithm f_{pro}

5. Rescheduling algorithm f_{re}

In the general case of the proactive-reactive scheduling algorithm, a schedule S_{pre} for the problem instance $i \in I$ is initially created using f_{pro}. The pre-schedule is created such that it optimizes the objective function g_{pro}. Then, the problem instance i_D is created by disturbing i with a disturbance $d \in D$. Finally, the rescheduling algorithm is used to create a reschedule for the disturbed instance i_D, using the objective function g_{re}.

Is simulated annealing local search TUSP solver suitable to create desirable reschedules?

A total of four different approaches based on the simulated local search TUSP solver have been proposed: partial rescheduling, guided partial rescheduling, schedule repair and guided schedule repair. The four approaches differ from each other in how they address the optimization of similarity of the reschedule with respect to the pre-schedule. The guided schedule repair performs the best out of the four proposed methods. This method addresses the optimization of similarity measures by using the pre-schedule as initial guess and then guiding the local search by including the similarity metrics in the objective function. The resulting schedules are both feasible and similar.

It can not be concluded that the reschedules generated using the proposed algorithm are desirable, due to the fact that it has not yet been determined whether the reschedules obtained using the proposed strategies are indeed regarded as similar to the pre-schedules. In this research, a specific similarity measure has been proposed for comparison between the pre-schedule and reschedule, but further research is needed to verify whether reschedules created according to this measure are indeed regarded as desirable.

While it can not be concluded that the reschedules are desirable, it can be concluded that the simulated annealing TUSP solver is suitable for finding desirable reschedules, as long as the desirability of a reschedule can be expressed as an objective function.

Which property of the TUSP pre-schedules must be improved in order to improve temporal flexibility?

A certain property of the pre-schedules has been identified, that influences the temporal flexibility of those pre-schedules. The slack between the arrival of a train unit and the starting time of the execution of its service task has been proven to have a strong negative correlation with the task lateness incurred due to rescheduling. This property can be used as a base for the proactive schedule generation algorithm.

8.2. Future Work

Both the rescheduling methods devised in this research as well as the presented abstract framework can serve as base research for extension by further research. Four main areas for extension by further research are identified: the quality of the disturbance model, the abstract framework, the significance of desirability of a schedule and the implementation of proactive rescheduling algorithms. Those four topics are described in the following sections.

8.2.1. Abstract Framework Extension

The abstract framework defined in this research is defined as an abstract one. It is not limited to the train unit shunting problem. In this research, however, the main focus has been the TUSP. For this reason, the implementations of the framework have only been described in the context of the TUSP. Further research could extend the scope in which the implementations of the framework have been defined.

8.2.2. Disturbance Model

In this research, the disturbance model has been limited to a single train arriving late. This could be extended in two main ways. Firstly, the type of the disturbance can be altered. The rescheduling algorithm can be extended to handle other types of disturbances, such as unexpected train arrivals. Further research could be used to analyze how flexible the rescheduling approaches are for dealing with other types of disturbances.

Secondly, the disturbance model can be extended by allowing more than a single disturbance to occur. It is possible that a breakdown on the railways causes more than a single train to arrive late. Further research could be performed in order to determine a suitable method for dealing with multiple disturbances. A possible way to do so is the consecutive use of the proposed rescheduling algorithm for each disturbance that occurs in order. Further research could be performed to determine whether this approach is viable and if the rescheduling performance degrades with the repetitive use of the rescheduling algorithm. Furthermore, when dealing with multiple disturbances, it is possible to obtain multiple reschedules. Further research could be performed to determine how the desirability should be calculated, with respect to the pre-schedule or the last known reschedule.

8.2.3. Desirability in Practice

A dissimilarity function has been used as a measure for desirability of a reschedule. A reschedule with a low dissimilarity value is assumed to be desirable, but this claim should be tested in practice. Furthermore, more research can be done in order to determine what exactly makes reschedules desirable and how those properties can be expressed as an objective function.

8.2.4. Proactive Scheduling Algorithm

In this research, a certain property of the pre-schedules has been identified, that strongly correlates with the temporal flexibility of the pre-schedules. This property is the slack between the arrival of a train unit and the starting time of the execution of the service task. It has been shown that the correlation exists, but no attempt has yet been made to implement a pre-schedule generation algorithm that optimizes for this property. Further research is needed to determine how this property can be optimized in a pre-schedule generation algorithm.

Bibliography

[1] Sporenplanonline. URL http://sporenplan.nl/.

[2] Shiwei Bao. A robust solution to train shunting using decision trees. Master's thesis, Delft University of Technology, 9 2018.

[3] Dimitris Bertsimas, John Tsitsiklis, et al. Simulated annealing. *Statistical science*, 8(1):10–15, 1993.

[4] Richard Freling, Ramon M. Lentink, Leo G. Kroon, and Dennis Huisman. Shunting of passenger train units in a railway station. *Transportation Science*, 39(2):261–272, 2005. ISSN 00411655, 15265447. URL http://www.jstor.org/stable/25769246.

[5] Willy Herroelen and Roel Leus. Robust and reactive project scheduling: A review and classification of procedures. *Katholieke Universiteit Leuven, Open Access publications from Katholieke Universiteit Leuven*, 42, 04 2004. doi: 10.1080/00207540310001638055.

[6] Willy Herroelen and Roel Leus. Project scheduling under uncertainty: Survey and research potentials. *European Journal of Operational Research*, 165(2):289 – 306, 2005. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2004.04.002. URL http://www.sciencedirect.com/science/article/pii/S0377221704002401. Project Management and Scheduling.

[7] Mikkel T. Jensen. Improving robustness and flexibility of tardiness and total flow-time job shops using robustness measures. *Applied Soft Computing*, 1(1):35 – 52, 2001. ISSN 1568-4946. doi: https://doi.org/10.1016/S1568-4946(01)00005-9. URL http://www.sciencedirect.com/science/article/pii/S1568494601000059.

[8] Esmee Kleine. Generating robust solutions for the train unit shunting problem under uncertainty: A local search based approach. Master's thesis, Eindhoven University of Technology, 9 2019.

[9] V. Jorge Leon, S. Wu, and Robert Storer. Robustness measures and robust scheduling for job shops. *Iie Transactions*, 26:32–43, 09 1994. doi: 10.1080/07408179408966626.

[10] V. JORGE LEON, S. DAVID WU, and ROBERT H. STORER. Robustness measures and robust scheduling for job shops. *IIE Transactions*, 26(5):32–43, 1994. doi: 10.1080/07408179408966626. URL https://doi.org/10.1080/07408179408966626.

[11] Roel Leus and Willy Herroelen. The complexity of machine scheduling for stability with a single disrupted job. *Operations Research Letters*, 33(2):151 – 156, 2005. ISSN 0167-6377. doi: https://doi.org/10.1016/j.orl.2004.04.008. URL http://www.sciencedirect.com/science/article/pii/S0167637704000641.

[12] VI Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707, 1966.

[13] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, USA, 2008. ISBN 0521865719.

[14] I. Sabuncuoglu and S. Goren. Hedging production schedules against uncertainty in manufacturing environment with a review of robustness and stability research. *International Journal of Computer Integrated Manufacturing*, 22(2):138–157, January 2009. doi: 10.1080/09511920802209033. URL https://doi.org/10.1080/09511920802209033.

[15] Roel van den Broek. Train shunting and service scheduling: an integrated local search approach. 2016.

[16] Roel van den Broek, Han Hoogeveen, and Marjan van den Akker. How to Measure the Robustness of Shunting Plans. In Ralf Borndörfer and Sabine Storandt, editors, *18th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2018)*, volume 65 of *OpenAccess Series in Informatics (OASIcs)*, pages 3:1–3:13, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN 978-3-95977-096-5. doi: 10.4230/OASIcs.ATMOS.2018.3. URL http://drops.dagstuhl.de/opus/volltexte/2018/9708.

[17] Peter J. M. van Laarhoven and Emile H. L. Aarts. Simulated annealing. In *Simulated Annealing: Theory and Applications*, pages 7–15. Springer Netherlands, 1987. doi: 10.1007/978-94-015-7744-1_2. URL https://doi.org/10.1007/978-94-015-7744-1_2.

[18] Guilherme E. Vieira, Jeffrey W. Herrmann, and Edward Lin. *Journal of Scheduling*, 6(1): 39–62, 2003. doi: 10.1023/a:1022235519958. URL https://doi.org/10.1023/a:1022235519958.

[19] S.David Wu, Robert H. Storer, and Chang Pei-Chann. One-machine rescheduling heuristics with efficiency and stability as criteria. *Computers & Operations Research*, 20(1):1 – 14, 1993. ISSN 0305-0548. doi: https://doi.org/10.1016/0305-0548(93)90091-V. URL http://www.sciencedirect.com/science/article/pii/030505489390091V.