Utrecht University

Master Thesis

Solving shunting yard disturbances using conflict-based search

First supervisor:
Dr. J.A. (Han) Hoogeveen

Second supervisor:
Dr. Ir. J.M. (Marjan) van den Akker

Author:
J. (Jochem) Brouwer

Supervisor NS:
Dr. R.W. (Roel) van den Broek

June 11, 2023

Utrecht University

Abstract

At railway shunting yards, trains are parked and service tasks are performed. Ongoing research has resulted in a local search algorithm to generate shunting plans which is capable of solving realistic, large problems. However, in practice, there might be disturbances at the shunting yard, which might make the computed schedules infeasible. In this thesis, we explore solving these disturbed problems by using *conflict-based directed search*. We take the original shunting plan, incorporate the disturbance, and then look at the conflicts which might arise. To solve this, we consider specific changes to the schedule which might resolve these conflicts. The conflict-based directed search algorithm is tested on two different shunting yards and is compared against the original algorithm which was used to generate the feasible, undisturbed shunting plans. We motivate the considered changes to the shunting plan in order to solve the disturbed problem and show how these could solve certain types of conflicts.

Contents

1 Introduction and Background **2**

2 Overview of HIP algorithm **4**
2.1 Train types 4
2.2 Subproblems 5
2.3 Train activities 5
2.4 Shunting yard overview 6
2.5 Overview of HIP 7
2.6 Evaluating the POS 14

3 Solving the shunting problem **15**
3.1 Conflict costs 16
3.2 The shunting problem with disturbances 16
3.3 Algorithmic approach 17
3.4 Research questions 17

4 Literature review **18**

5 Conflict-based directed search algorithm **20**
5.1 Iterating the neighbourhoods 20
5.2 Neighbourhoods considered per conflict 22
5.2.1 Crossing 22
5.2.2 Track length violation 30
5.2.3 Arrival Delay 34
5.2.4 Departure delay 37

6 Experimental setup **43**
6.1 Generating the problems 43
6.2 Running the experiments 44

7 Results **45**
7.1 Unresolved problems 48

8 Conclusion **49**

9 Discussion and further research **50**

10 References **51**

1 Introduction and Background

Nederlandse Spoorwegen (NS / Dutch Railways) is the main corporation that uses the Dutch railway network to transport passengers. There are several other (smaller) companies which usually do more local transport. The Dutch railway network is one of the busiest in the world. The peak demand in terms of passengers on these trains is in the morning rush and in the evening rush. At those times, almost all trains are actively driven. However, outside the rush hours, it does not make sense to use mostly empty trains, and therefore those trains are parked at a shunting yard.

At a shunting yard operations take place to ensure that the timetable of the day can be executed. This means that some trains have to be split into multiple smaller trains, whereas others have to be combined to form longer trains. Besides splitting and combining, there are also service tasks which have to be executed. For instance, mandatory technical inspections have to take place from time to time, and the trains have to be cleaned outside and inside as well.

Outside these rush hours, the shunting yards are very full. Because of that, planning the movements of these trains and scheduling the service tasks is not trivial. Therefore, a *shunting plan* has to be created. Since the timetable (including the rolling stock) of NS is known long beforehand, it is possible to create a *shunting plan*. The shunting plan describes the movements of each train at the shunting yard, such as splitting, coupling and executing service tasks. Currently, this is all done by hand, but the R&D department of NS is working on an algorithm to help creating these shunting plans, called the *Hybride Integrale Planmethode* (HIP: Hybrid Integrated Planning method). This is not yet used in practice, but it manages to solve realistic problems, where sometimes there are only one or two conflicts left which HIP is unable to solve, but which human planners can relatively easily solve.

However, in reality, there might be disturbances, such as trains arriving earlier or later than planned. This could make the original shunting plan infeasible. For instance, if a train arrives 5 minutes later, it could be the case that a track which the train should be driving through according to the shunting plan is now occupied by another train. These disturbances are solved ad-hoc, but it would be helpful to extend the algorithm in order to help solving these disturbances as well. Since the original shunting plan is known, but there are now conflicts, it is hopefully possible to solve these conflicts by editing the original plan such that the disturbance is taken care of.

This thesis will focus on solving this disturbance problem. The original HIP algorithm is extended in order to solve these disturbances. The end goal is to create an algorithm which outperforms the original HIP algorithm in these small disturbances: it should solve more situations faster than HIP solves it. The key idea behind the algorithm is to perform logical changes, based on the conflicts which arise.

In Chapter 2, the HIP model to analyse this problem is reviewed. In Chapter 3, we show how HIP solves the shunting problems. In Chapter 4, a short

literature overview is given. In Chapter 5, the algorithm to solve disturbances is described. In Chapter 6, we describe the experimental setup. In Chapter 7, we show the results of this experiment. In Chapter 8, we conclude this thesis. In Chapter 9, we finally do a short discussion and point to areas which would be interesting to research further.

2 Overview of HIP algorithm

This section gives an introduction to the relevant parts of HIP. The model which HIP uses is re-used in this thesis. The algorithm we designed works on the data structure which HIP uses, so if the algorithm outperforms HIP then it can also be integrated in HIP, in order to improve it.

As stated earlier, the Dutch railway network is used mainly by NS, but it is also used by regional companies. These will also need to use the shunting yard. In practice, these companies get assigned shunting yard tracks for their trains to use, so NS has to only consider a subset of the shunting yard tracks.

The arriving trains for a shunting yard problem are given as input with these parameters: their arrival time, their arrival track and the composition of the train. The composition is known exactly: it is known what specific train numbers arrive and in which order. A specific member of the train, thus identified by its unique number, is called a *train unit*. Due to the unique numbers of these train units, a specific train unit can thus be distinguished from other units.

The departing trains are also given as input with almost the same parameters: the departing track, the departure time, and the departure *composition*. The difference in this input is that the exact train units of this train composition are not defined: only the train *type* and the order of these types are given.

For each train unit, a list of service tasks is also given as input. These are for instance cleaning (inside or outside), or technical inspections. Some of these tasks can only be done at certain tracks, because certain facilities are only available there. For instance, for outside cleaning, the train unit moves through a track with brushes at the side.

2.1 Train types

A *train* is a (possible) combination of multiple *train unit(s)*. These train units are of a certain *train type*, where train units of the same *type* can be combined to form a longer *train*. These specific train units can also be classified into several *train sub-types*, where the *train sub-type* usually represents the length of the train unit, i.e. the number of carriages in the train.

For instance, a train type could be VIRM, which has two sub-types: VIRM4 and VIRM6. VIRM4 has 4 carriages and VIRM6 has 6 carriages. It is possible to combine trains of type VIRM, thus for instance combining two VIRM4 units, or a VIRM4 unit with a VIRM6 unit. Besides VIRM, there are also other types, such as the SLT, the ICM and the FLIRT. Different train types cannot be coupled to each other.

NS mostly uses electric multiple unit (EMU) train units. These train units are self-propelled and thus do not need a locomotive in front to successfully execute a journey. They also have a cab in both directions of the train, thus it is not a problem if the train has to reverse its direction. EMU train units consist of multiple carriages and these are in practice never split. However, multiple EMU units can be joined together to form a longer train.

There is one exception: the Intercity Direct line between Amsterdam and Breda uses units where on each side of the train, a locomotive is placed, which can thus be uncoupled from the carriages and thus form an arbitrary length train. However, in practice, these trains either consist of 7 or 9 carriages with a locomotive in front and at the back of the train. These are in practice also never split.

2.2 Subproblems

The problem which HIP tries to solve consists of multiple subproblems:

- Each departing train should depart on time, in the right composition
  - This means that each arriving train unit should be matched to a departing train
  - It could be possible that an arriving train has to be split/combined on the yard in order to reach the desired departure composition
- Each train unit should execute their service tasks
- Each train unit should move around the shunting yard and be parked during its stay

In order to solve this problem, a shunting plan has to be created, which denotes what train activities are done, at which track, and at which time. The shunting plan also includes the routes which the trains take to drive over the shunting yard.

2.3 Train activities

In order to create a shunting plan such that the timetable can be executed, multiple different activities can be assigned to a train unit:

Simple movement

This consists of the train moving in forward direction (based upon which side the driver sits) to a new location. It is not allowed to drive the train backwards. It might be that the driver has to control switches manually. The driver has to visually check whether or not the train can be moved to a new track. It is not allowed to occupy a track which has a train actively driving on it.

Reversal

This consists of the driver hopping off the train, walking alongside the train to the other side, and taking control at this side. This effectively reverses the train and still complies with the rule that trains are only allowed to move forward. This reversal movement is also called a *saw movement*.

Coupling

This consists of a parked train coupling with a driving train of the same subtype. It is not possible to couple train units which are of different type. For instance, a SLT4 and SLT6 can be coupled (their type is SLT, and their subtypes are thus SLT4 and SLT6), but a SLT4 cannot be coupled to a VIRM6.

Splitting

This consists of decoupling/splitting a parked train into two trains. (Usually two train units).

Service

During the service period, the train is parked at a service platform and the service task is being executed. Services have a certain *frequency* and in some cases a *maximum interval*. It is desired that each service is executed at a rate of the frequency. However, if this is not possible in practice and the maximum interval is defined, then that service task should be done before the maximum interval expires. For instance, train cleaning should be done daily, but if necessary, it can be skipped. However, train technical inspections should be done at a certain frequency and must be done before the maximum interval is reached, otherwise it is not allowed to participate in the network due to safety regulations. For each train unit, a list of service tasks is included as input to the problem, and these must be executed, so skipping these in HIP is not possible.

2.4 Shunting yard overview

Shunting yards consist of a set of tracks and switches. Some service tasks are only allowed to be performed on a certain track. For instance, cleaning the trains is done at cleaning platforms. The cleaning of the train outside is done using a washing facility, similar to how a car wash works. For technical tasks, it is sometimes necessary to inspect the bottom of the train or the top: certain facilities are necessary for that.

Some shunting yards have tracks which end in a buffer. These tracks are thus last-in-first-out (LIFO). Shunting yards consisting mostly of those tracks are called *shuffleboard* yards; see Figure 2. In case that a shunting yard consists of a lot of tracks which can be entered from both sides, then it is called a *carousel*; see Figure 1.

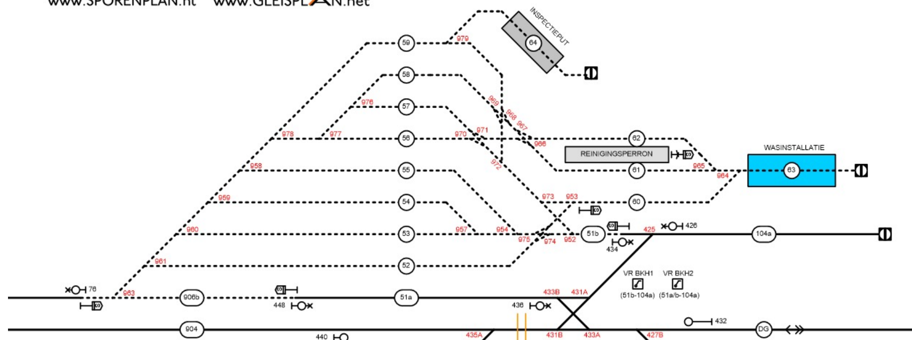

Figure 1: Kleine Binckhorst, a carousel-type shunting yard. Track 904 is a mainline track, thus used for passing trains (trains which carry commuters), and is thus not part of the yard. The track right above track 904 (which is unlabeled in the picture, but this is track 906a) and track 104a are entrance/departure tracks to the yard. Copyright: sporenplan.nl

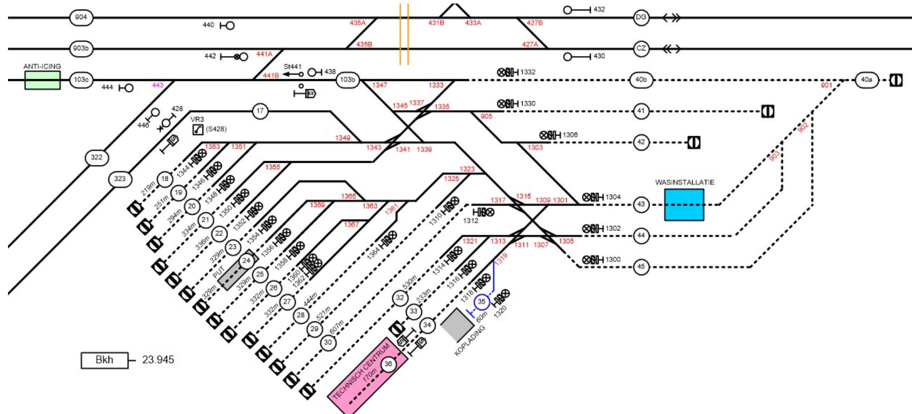

Figure 2: Grote Binckhorst, a shuffleboard-type shunting yard. Here, track 103b and 17 are used to enter/depart the yard. Notice that track 904 is visible as well, so this yard is very close to Kleine Binckhorst. Copyright: sporenplan.nl

2.5 Overview of HIP

NS is thus developing an algorithm to help creating feasible shunting plans, called HIP. This algorithm was originally designed by Roel van den Broek for his Master Thesis, “Train Shunting and Service Scheduling: an integrated local search approach” [Bro16]. HIP makes use of Local Search in order to find a

feasible shunting plan. This section explains how the relevant parts of HIP work.

The input to HIP consists of:

- Arrival trains
  - Arrival time
  - Arrival track
  - Train composition including train numbers
  - List of service tasks per train unit
- Departing trains
  - Departing time
  - Departing track
  - Train composition (only train types)
- Shunting yard
  - Tracks
  - Switches
    - Switch type (which tracks can be reached from each side of the switch)
  - Service facilities
    - Track
    - List of service types which can be served
- Service tasks
  - Duration per train type

In order to solve the shunting plan, HIP uses Local Search. In order for this to work, the shunting plan should be modelled. HIP models this using a *Partial Order Schedule* (POS). A POS is feasible if there are no conflict costs. These conflict costs arise if the POS is evaluated and conflicts are found, for instance if a train leaves the shunting yard too late.

The POS is a graph consisting of nodes, which are activities, and arcs, which are precedence relations. For a certain activity to start, it is therefore necessary that all activities which point to this activity have finished. Due to this, the graph is therefore a directed acyclic graph, otherwise no times can be scheduled to the activities.

We will give a simplified explanation of the POS graph in order to explain the relevant elements for our problem.

The POS can be thought of a graph consisting of two node types: either a movement, or a parking operation. There are special movements to be considered here: if the movement is from a gate (an entrance/exit track to/from

the shunting yard) to the yard, then a new train arrived. If the movement is from the yard to a gate, the train departs. A parking activity can either be a simple parking activity, or it could be at a service track where the service is being executed as well. In this graph, each movement activity is connected to one or more parking activities, and each parking activity is connected to a movement activity. We shall see later that if a movement activity is connected to more than one parking activity, this is used to model either a split or combine operation.

It should be mentioned, that in the POS movement nodes do not have explicit starting/ending times and an exact route (only their origin and destination, and the train unit(s) affected). For parking activities, starting and ending times are not assigned either. We will show later how these routes and times are computed.

The arcs between these nodes thus represent the precedence relations. All previous tasks have to be performed, before the task at the current node can start.

The model used to represent a shunting problem thus takes into account the movements of the trains, the service activities, and split/combine operations. It does not take into account other factors which are relevant in practice: for instance, it does not take into account schedules for the drivers. It is therefore assumed that there are enough available drivers at all times in order to execute the schedule.

In an example below we will show how this model can be used to model all the mentioned activities on the shunting yard, including combining and splitting. We will present both the POS (in our simplified representation) and a visualization of what happens at the shunting yard.

Example problem

We first give the shunting yard layout and the inputs to the problem.

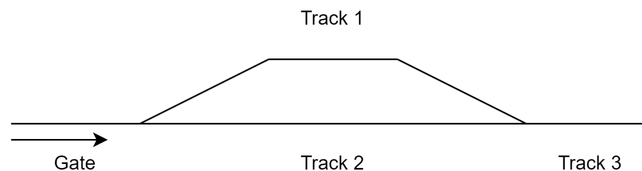

Figure 3: The shunting yard layout. Cleaning tasks can only be done at track 1.

Arrivals

Time	Type (name)
10:00	SLT-4 (A) - SLT-4 (B)
10:10	SLT-6 (C)

Departures

Time	Type
13:00	SLT-4
14:00	SLT-6 - SLT-4

Service tasks

Train name	Task (time)
SLT-4 (A)	Cleaning (20 minutes)
SLT-6 (C)	Cleaning (30 minutes)

Below, we give the POS which, upon evaluating, leads to no conflicts and thus solves this problem.

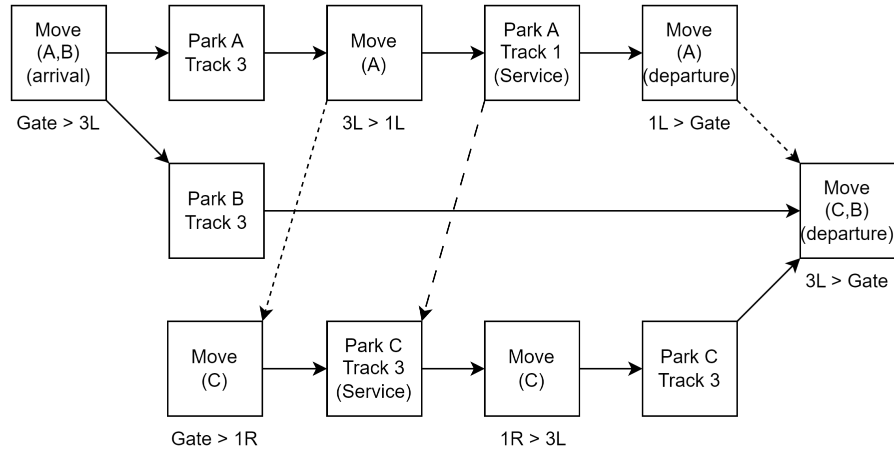

Figure 4: The POS. There are three types of precedence relations here: either an activity has matching train unit(s) (full lines), uses the same infrastructure (dashed), or uses the same service facility (long dashes). The X > Y texts under the movement nodes describe the origin track (X) and the destination track (Y) of the movement. These tracks are followed by either L (left) or R (right) to describe from which side the train moves on the track.

We will show below what happens on the yard:

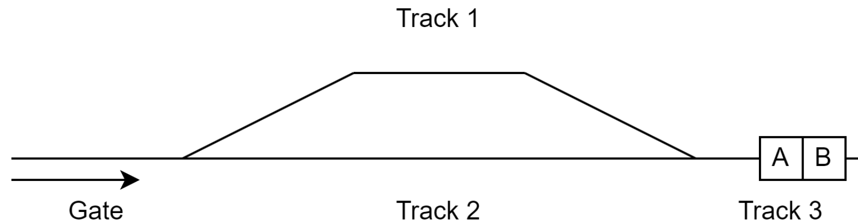

Figure 5: The train *(A,B)* arrives at the gate and moves to track 3.

Notice, that after this movement node, this movement node now points to two parking activities. This models the split operation in our model.

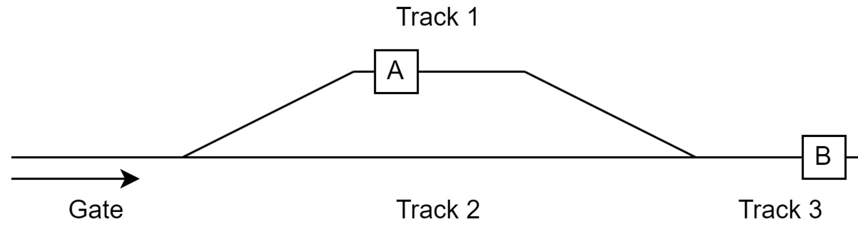

Figure 6: Train unit A moves from track 3 to track 1.

As we will see in a later section, the earliest starting time for each activity is calculated. So, if all precedence relations have been assigned a time, then the maximum end time of these relations is thus the earliest start time to start a task. Hence, we can immediately start performing the service task for train unit A on track 1.

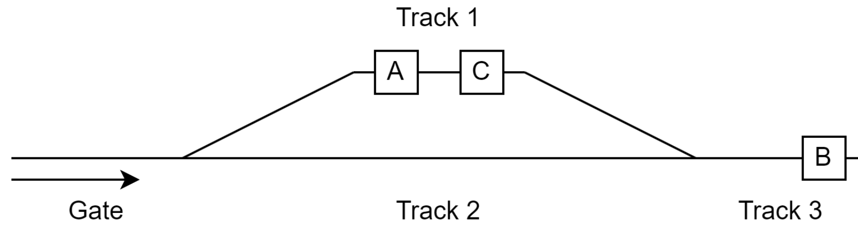

Figure 7: Train unit C arrives at the shunting yard and moves to track 1. Note, that it thus performs a saw movement on track 3. Before we can start the service task for train unit C, the service task for train unit A has to finish first.

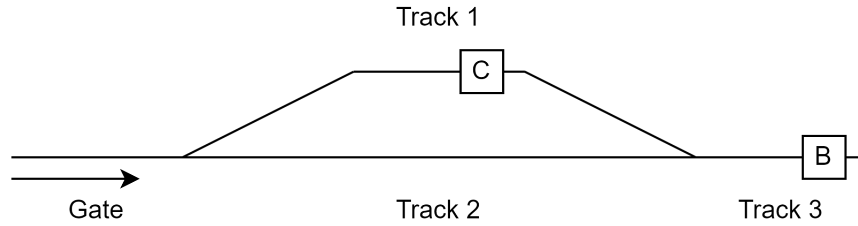

Figure 8: Train unit A departs from the shunting yard. Depending upon the time assignments of the previous actions, C will either have finished the service task, is currently doing it, or just started it.

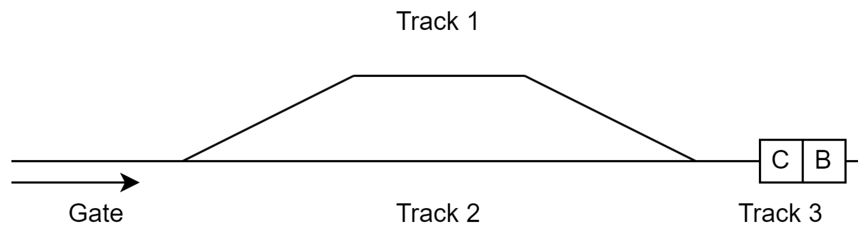

Figure 9: Train unit C moves to track 3. Notice, in the POS, that the next movement joins two parking tasks together, so this is a combine action as well. Train units C and B are now combined.

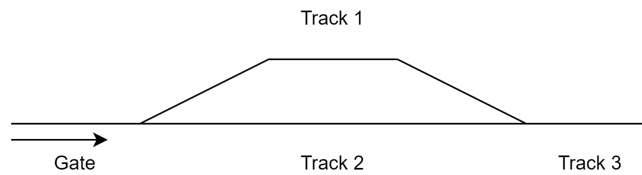

Figure 10: Finally, train *(C,B)* leave the shunting yard.

Notice that the order of the trains departing from the yard is respected: if the train would leave in order *(B,C)* this would not match the input.

2.6 Evaluating the POS

In this section, we describe how the POS is evaluated. In order to evaluate this, HIP has an ordered list of movement activities, which is the order in which these movement activities are considered. However, we have seen before that there are also precedence relations between matching infrastructures, or the order of service tasks at a facility. HIP will choose the order to evaluate those (which is deterministic). There are two steps: first, we calculate the routes, and then we calculate the starting times. In this section we will also briefly mention some of the conflicts, which we will describe in-depth later.

Computing the route and possible conflicts

We thus have an ordered list of movement activities. These are considered sequentially. For each movement, there is thus only one train moving. We can thus formulate a shortest-path problem, where the costs are the time the train takes to move from the origin to the destination. There are also situations, when calculating the route, the train **must** move through another train. In this case, there is a conflict (a crossing), and we will penalize this by incurring a very large cost in order to ensure that these crossings do not occur if these are not necessary (thus if the algorithm has to choose between a very long route, or a short route where it crosses a train, it will still choose the long route). The train could now move to a track, which upon arrival, now has train units on the track which have a combined length larger than the total track length (the train units thus do not fit on this track). This is tracked as well, because this is a track length violation conflict.

Computing the starting time

In a real world scenario, there is no requirement that only one train is moving at a specific moment in time at the shunting yard. However, the movements are still subject to safety rules, for instance a minimum amount of time before a train is allowed to use a track which another train just used as well. In order to calculate the starting time, the earliest time is calculated at which this is possible. Note, that in order to do so, all the precedence relations (such as service tasks or previous movements by the train) have to be finished. It is now also possible that multiple trains drive at the same time. However, it is not possible that these trains use a same piece of infrastructure simultaneously while they are moving. Therefore, once a train starts driving a certain route, then that entire route is blocked for the time the train is driving. In order to calculate the earliest starting time, it is also not possible that the train crosses other parked trains or violates the total parking capacity of the destination track. This requirement ensures that no more conflicts are introduced in this step. The starting time of the movement is then set to the earliest possible starting time of that movement. HIP will always choose the shortest route available when evaluating the POS.

3 Solving the shunting problem

In the previous section we have described how the POS is evaluated, and we have given an example POS which is a valid shunting plan, without any conflicts. The goal of HIP is, when given the described inputs, to find a valid shunting plan. HIP uses local search to solve this problem. In local search, random changes are made to the POS and then the plan is re-evaluated. Upon re-evaluation, conflicts might arise, which we will describe in this section. The goal of HIP is to find a feasible shunting plan: a plan with zero conflicts.

In this thesis, we will take a feasible shunting plan (generated by HIP) and then introduce a disturbance, where one of the arriving trains arrives at the gate at a later time. We will then re-evaluate the POS, which might introduce conflicts and thus an infeasible shunting plan. Note that these conflicts are thus caused by the delayed train. For instance, due to the delayed train now blocking a route at a later time, other trains might start their movements later as well, thus delaying those train movements, which might cause extra conflicts. We will now discuss the types of conflicts which might arise upon re-evaluating the POS when we have incorporated this disturbance.

Arrival delay

In this conflict, the conflicted train is scheduled to enter the shunting yard at a certain time, but this is not possible and therefore the train has to enter the shunting yard later than planned. This train could either be the disturbed train, or be a victim of some disturbance which happened earlier. This is a conflict, because in reality this would mean that a train is parked at the gate of a shunting yard, which practically means that the train has to park outside the shunting yard and is therefore blocking the railway, which is normally used by passing trains. These conflicts usually arise because the route from the gate to a destination track is blocked (there is another train which uses part of the infrastructure of the route, which thus blocks access to all these infrastructures).

Departure delay

This is the same as an arrival delay, except the train now leaves the shunting yard later than scheduled. This could be because the route to the gateway is blocked, but it could also be the case that upon evaluating the disturbed POS, service tasks or movements are scheduled too late, which means the train spends too much time at the yard and thus can only leave later than the scheduled time.

Crossing

When calculating the routes of the trains, crossings of trains are avoided. However, in some cases, it is not possible to get from the origin to the destination without crossing a train, for example if the train is at a LIFO track where it has to move through a train which entered the LIFO track later. When calculating

the route, the amount of crossings is minimized, but in some cases it is thus not possible to get to the destination without crossing another train.

Track length violation

This conflict arises when the total length of all trains parked at a track is longer than the track itself. The capacity of the track is therefore not sufficient to allow the trains to park there.

3.1 Conflict costs

When HIP evaluates the POS and it finds conflicts, it will also calculate a conflict cost, in order to distinguish between two instances of the POS, in such a way that it can be determined which one is the "best" POS (close to a feasible solution). If there are no conflicts, the cost is 0. Each conflict is multiplied by a weight in order to be able to punish certain conflicts more over the other.

Arrival delay

The cost of this conflict is the total delay time, multiplied by the arrival delay weight.

Departure delay

The cost of this conflict is the total delay time, multiplied by the departure delay weight.

Crossing

The cost of this conflict is equal to the crossing conflict weight.

Track length violation

The cost of this conflict is the extra track length (which thus does not physically exist) used, multiplied by the time this extra track length is used, multiplied by the track length violation weight.

Note, that thus all these conflicts have variable costs (depending on the duration of the conflict, and in case of the track length violation also the size of the track length violation), except the crossing conflict which has constant costs.

3.2 The shunting problem with disturbances

In this section, we describe the shunting problem with a disturbance. In this problem, a feasible POS is given. This problem models a realistic situation. A *disturbed train* is a train which is scheduled to arrive at time $\mathbf{X}$ at the shunting yard, but now it arrives at $\mathbf{X} + \mathbf{T}$. The problem is solved if a new, feasible

plan is created subject to the constraints of the original shunting yard problem. Additionally, it is allowed to edit the original shunting plan from time $\mathbf{X} - \mathbf{A}$ onwards; all actions before this time are not allowed to change.

Hence, in reality, this would happen: at time $\mathbf{X} - \mathbf{A}$, the shunting yard operator gets a notification that the *disturbed train* now arrives at $\mathbf{X} + \mathbf{T}$. The shunting yard operator can now respond by editing operations at that time, if it is found out that the original shunting plan is now infeasible with this disturbance.

3.3 Algorithmic approach

Whereas HIP will try to solve this disturbed problem by trying random operations, the key idea in this thesis is to focus on the conflicts which have emerged and try to perform logical operations in order to see if these conflicts can be solved. We call this *conflict-based directed search*. Per conflict it makes sense to try to solve the conflict in a natural way: for example, if there is a track length violation, it makes sense to move at least one of the trains on the track to another track. This should hypothetically outperform HIP, since it performs actions which directly have to do with the conflict itself, instead of trying random actions. The reason why we make this statement can be explained with a trivial example. Take for instance a shunting yard problem where there are essentially two shunting yards (connected to the main railway network). For instance, we take a shunting problem involving both the Kleine Binckhorst and the Grote Binckhorst. If a problem arises on Kleine Binckhorst, then HIP could try to change the POS at the Grote Binckhorst, while *conflict-based directed search* would take a local look and only try to change things at the Kleine Binckhorst. A full overview of our algorithm is given in Chapter 5.

3.4 Research questions

The goal of this thesis is to find an answer to the following questions:

- Is the conflict-based directed search method a feasible way to solve the disturbance problem at a shunting yard?
- How does this method compare to the HIP local search algorithm?

4 Literature review

Research of train shunting problems is a rather new field in operations research. The Train Unit Shunting Problem (TUSP) was defined by Freling et al. in 2005 [Fre+05]. This original problem only concerns parking and shunting the trains and does not consider service tasks. In a PhD thesis, Lentink [Len06] has shown that this problem is NP-hard. The service tasks were introduced later by van den Broek [Bro16]. That thesis formed the basis for the work at NS to generate shunting plans for realistic situations, thus also including the service tasks. The task at hand in this thesis was to calculate the capacity of the shunting yard: what is the highest amount of train units on the shunting yard which still leads to feasible solutions? The work used simulated annealing, which is a local search heuristic where changes to the current solution which make the solution worse can be accepted, based upon a chance which depends on the current "temperature": at higher temperatures the chance that a worse solution gets accepted is higher than at a lower temperature. During the search, the temperature gradually decreases.

The thesis was later followed by [Bro+22], which had as goal to generate feasible shunting plans for realistic problems. [Bro16] had already shown that it was possible for a large instance to generate a feasible shunting plan, thus this signalled that this model could also work on large, realistic problems. In [Bro+22], the local search model is explored further and it is shown that it is capable of solving real-world instances for the first time.

In this thesis, we focus on the shunting problem with service tasks. However, there are other activities at the shunting yard which are also important to consider. For instance, the service tasks themselves need mechanics in order to execute them. Some of these service tasks require that two mechanics are present at the train in order to execute the task. This therefore needs scheduling and synchronization: the two mechanics need to be at the same train at the same train. This is currently done by the mechanics ad-hoc. In [Sza23], scheduling and synchronizing the schedules for the mechanics is considered, where also train movements are taken into account. The approach also uses simulated annealing. Also, the robustness of the plans are evaluated. If trains arrive at a different time than scheduled at the shunting yard, this might make the schedules for the mechanics infeasible. The ability to recover from these disturbances are also evaluated.

Robust schedules for shunting yards is also a new field in research. If generated shunting plans are not robust, then any realistic change (such as trains arriving earlier or later) would make the schedule infeasible and also hard to recover from. In [BHA18], a method to measure the robustness of shunting plans is proposed. The robustness measures are then tested against a Monte Carlo simulation to see how correctly the measures score the robustness of the schedule.

There is not a lot of research about disturbances at shunting yards. However, disturbances on the railway network have been widely researched. In case of railway network disturbances, Törnquist Krasemann [Kra12] introduces a

greedy algorithm which uses depth-first search on an event model, which back-tracks in case the algorithm finds out that there is a deadlock due to track occupations. The algorithm should finish within 30 seconds and it is applied on real life situations. For a literature overview of disturbances and disruptions (large disturbances) in railway networks, see [Cac+14].

Research based on shunting yard disturbances is quite new. In [Ono20], the disturbances considered are trains arriving earlier or later, or that trains arrive in a different composition. The disturbance is solved using simulated annealing. Shunting plans are also compared to each other, to see how similar the solved disturbance plan is compared to the original shunting plan. To measure this, the driving plans of the machinists are considered. In [Ste20], four algorithms are proposed in order to solve the disturbance. There is heavy focus on the desirability of a shunting plan. This consists of plans which are desired in real life, such as minimal changes to a schedule in order to help human planners in case that there is a disturbance: if the plan changes minimally, then the plan would resemble a plan which is already known by the planners. There is also focus on the service tasks, where it is undesired if the service tasks schedule changes.

5 Conflict-based directed search algorithm

In this chapter, we describe the algorithm used in order to solve the shunting problem with disturbances.

The key idea of the algorithm is to take each conflict and make local changes in order to solve this conflict. This is done by taking neighbourhoods (this is a set of operations to apply on the POS) and applying them to relevant parts of the conflict. For instance, if one train crosses another train, then it makes sense to only perform operations on one of these two trains. We first describe the neighbourhoods which are considered for each conflict, and then describe how we iterate over these neighbourhoods in order to edit the POS. We also describe what the operations change to the POS if the neighbourhood is considered. It should be mentioned that the order of the neighbourhoods evaluated does not matter: our algorithm first evaluates all neighbourhoods for all conflicts in the current POS and then picks the best. There is thus no priority of a certain neighbourhood over another.

5.1 Iterating the neighbourhoods

In this section we describe how exactly the neighbourhoods are iterated. This is done by creating a directed acyclic graph, with initially one root node: the POS with the disturbance built-in (and which thus has conflicts). Each node stores the following information:

- The order of operations to reach this node (the root node thus has an empty list of operations). This therefore describes how to change the POS in order to get the situation at this node.
- The depth of this node (how much operations are executed in order to reach this node).
- The conflict cost of this node, after executing the operations.
- Whether the node has been visited already.

Note that this graph thus starts with only one unvisited node. In order to perform an iteration, take the unvisited node with the lowest conflict costs. Now, for each of the neighbourhoods considered (we will describe exactly what neighbourhoods are evaluated per conflict later this chapter), evaluate all these neighbourhoods and per neighbourhood pick the operation which yields the lowest conflict costs. Note that a neighbourhood is a set of operations, and an operation is thus a neighbour in the neighbourhood. Thus, per considered conflict, we iterate over all neighbourhoods which thus contain a set of operations which might decrease the total conflict costs of the POS (or solve the problem). Per neighbourhood, we pick the best operation (which thus yields the lowest total conflict costs for that neighbourhood) and insert a new node which contains all operations needed to reach that node (thus the changes to the POS), the conflict costs, and the depth. This node is marked as unvisited. For the current

node, mark it as visited and now revert all operations used to reach this node, such that we are back at the root node. Now start the procedure over (so pick an unvisited node with the lowest costs, apply the operations in order to reach this node, and re-iterate over all neighbourhoods). Repeat this procedure until the conflict costs are zero, which thus leads to a feasible solution. We will give an example later.

In case that there are multiple nodes with the same conflict cost, a node is picked at random.

It is possible that for two operations, taken from different neighbourhoods, we end up with exactly the same POS. In order to prevent that we insert the same POS twice as unvisited node into the graph, we introduce a *Tabu list*. Each time a neighbourhood is evaluated, the resulting POS (which thus has the lowest conflict cost for that neighbourhood) is inserted into the Tabu list. When evaluating a neighbourhood, thus when iterating over all the possible operations of that neighbourhood, the resulting POS after applying the operation is checked to be in the Tabu list. If this is the case, then that operation is not considered. This thus prevents that the same POS is inserted as unvisited node into the graph twice. If we would do that, then this would result in evaluating the same starting point twice, which would thus lead to evaluating exactly the same operations twice.

This algorithm naturally also allows for backtracking, which we will describe in the following figure:

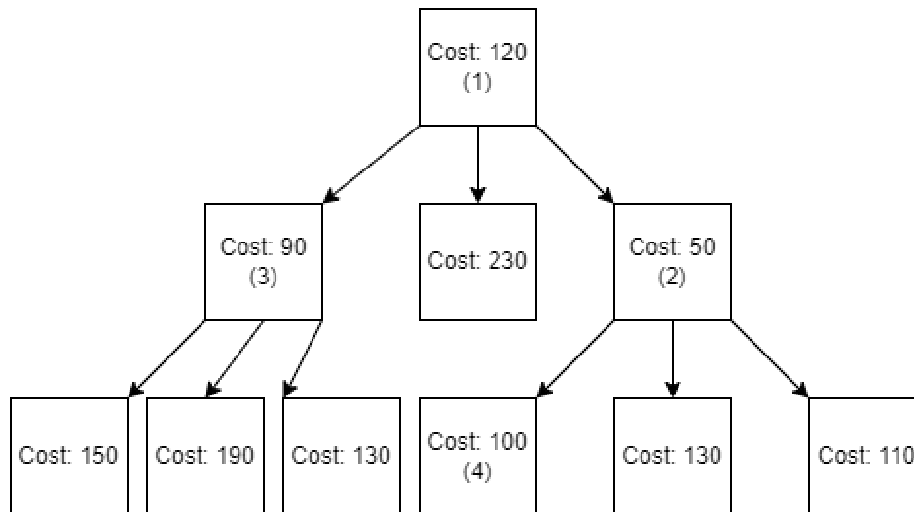

Figure 11: Example search. The numbers in a box describe at what point this candidate is considered.

The original conflict cost of this problem is 120. We start in node (1) and then generate the nodes associated with the considered conflicts. We thus consider three neighbourhoods here, and for each neighbourhood we thus pick the

best operation. Then, cost 50 (node (2)) is the lowest cost, thus we evaluate the corresponding solution to find the conflicts and again iterate over the neighbourhoods based upon the conflicts at that node, thus again picking the best operation for each of the considered neighbourhoods. However, in this case, it only generates higher conflict costs than the costs at node (2). If we now consider all unvisited nodes, we end up with node (3). Note that this effectively backtracks to the root node, and now performs a different operation than the operation we performed at node (2). We again generate the new nodes starting from node (3), and when again considering all unvisited nodes, we now end up at node (4). We thus backtrack again, first execute the operation at node (2), and then execute the operation at node (4). The search tree is thus evaluated in the order of best (lowest) cost first, for each of the unvisited nodes. Even if we cannot find a new solution which has lower conflict costs than the current lowest costs, we continue to generate new solutions.

5.2 Neighbourhoods considered per conflict

In this section, we describe for each of the four conflicts, what neighbourhoods we consider in order to solve the conflict. Note that per iteration of our algorithm, each conflict is considered and thus all operations for the considered neighbourhoods are evaluated per conflict. For some of these neighbourhoods, we will give an example situation of the conflict and then show how an operation in the neighbourhood solves the conflict. There are some situations, where other operations could be applied as well which would then also solve the problem.

Per conflict, there are thus multiple neighbourhoods considered. It is possible that, after evaluating the neighbourhoods, two or more resulting operations solve the same conflict. However, both these operations yield a different POS (this yields POS A and POS B). Our algorithm will always pick an unvisited node which has the lowest conflict costs at that point. Therefore, at some point (if the problem is not yet solved), both A and B would be considered as starting point for the next evaluation iteration of the algorithm.

We will not give an example to show how each operation could possibly solve a conflict. It is left to the reader to convince themselves that it is reasonable to explore the mentioned neighbourhoods. In our examples, the conflict is always solved. In practice, some operations yield a better conflict score but do not solve the conflict in consideration. For all conflicts except for crossing conflicts, reducing the duration of these conflicts decreases the conflict score, and is thus an improvement of the situation.

5.2.1 Crossing

Recall that a crossing is a conflict which happens if two trains drive through each other. This has an *active* train A and a *passive* train B. The active train is a moving train: the passive train is parked somewhere, and train A drives through it. This happens if a route for train A is calculated from the origin track to the target track of a movement activity: if the only possibility to reach the

target track is to drive through another train, then a crossing conflict happens. Figure 12 shows a situation where a crossing conflict occurs.

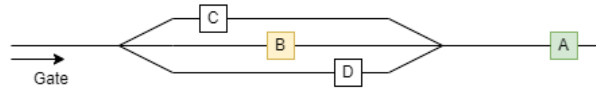

Figure 12: A problem where a crossing conflict occurs: A has to drive to the gate. Trains B, C, and D are parked trains.

These crossing conflicts are thus caused because it is not possible for the active train to reach the destination track without crossing through another train. Therefore, it makes sense to change target destinations of where the active train is moving to, or change the track where the passive train is staying.

Note that for these conflicts, the active train has two movements which are related: the movement which moves the train from some track to the origin track, and the movement which moves the train from the origin track to the destination track. Note that in the second movement, the conflict arises. In most operations, a movement activity is taken. For those operations, both these movements are considered. For instance, it would make sense to either change the origin track where the conflict arose (the train now departs from a different track), or we change the destination of the movement (the train now drives to a different track).

We will now explain which neighbourhoods are considered for this conflict. For each new section marked by a **bold** heading, we describe a new neighbourhood. This neighbourhood is thus a set of operations: we explain what the operation does and sometimes give an example to show how this changes the POS.

Parking change

In this neighbourhood, a movement is taken and the destination track is changed to any available track. In the figures below we give a simple situation and show how this operation solves the conflict.

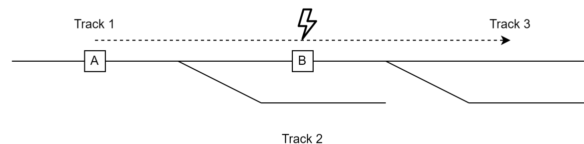

Figure 13: A problem where a crossing conflict occurs: train A has to drive from track 1 to track 3, thus through train B.

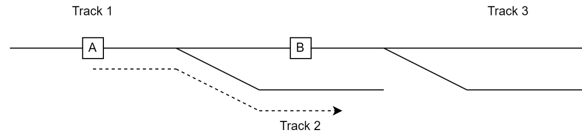

Figure 14: The trivial solution here is to park train A at track 2 instead of track 3.

Parking swap

In this neighbourhood, we take two movements: the movement of the active train A is taken, and the movement of the passive train B is taken. Note that the movement of the active train A is thus the movement where train A drives through train B at track O, causing the conflict. The movement taken for train B is the movement which starts at track O to another track. In this neighbourhood we swap the destination track of train A and train B. So, the original destination track of train A is now the destination track of train B, and vice versa.

In the figures below we show a situation where this operation solves the conflict.

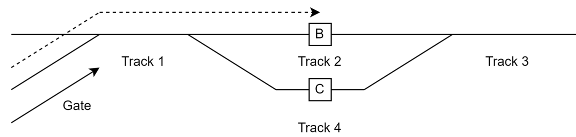

Figure 15: In the original situation, train B drives from the gate to track 2.

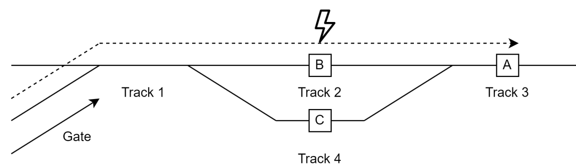

Figure 16: Then, train A drives from the gate to track 3. It thus has to cross either train B or C.

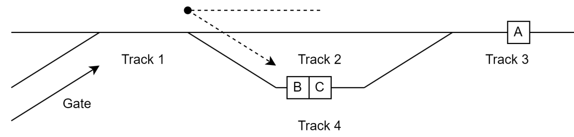

Figure 17: Then, train B drives from track 2 to track 4.

In order to solve this problem, we consider swapping the movements from Figure 16 and Figure 17. Therefore, instead of moving A to track 3, we now move it to track 4. Train B moves to track 3 instead of track 4. After the original movement from Figure 15, (park B at track 2) the following happens:

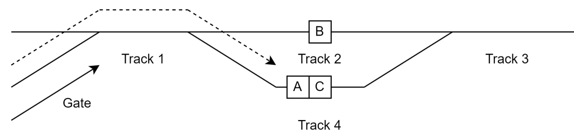

Figure 18: Instead of moving to track 3, train A moves to track 4.

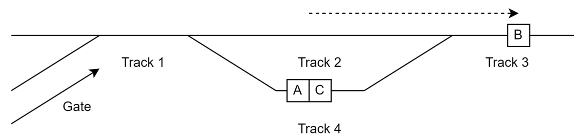

Figure 19: Then, train B moves from track 2 to track 3.

Parking insert

In this relatively complex neighbourhood, per operation a movement is considered. In this operation, only the second movement (so the movement from the origin O to the destination D, which caused the crossing) is considered, which we call movement A. In this operation, movement A changes its destination D to another track T. Another movement is now inserted, which has origin T and destination D. Note that this movement should be inserted in the ordered movement activities list somewhere between the original movement A and the movement which was originally after A (so the movement from track D to some other track, this is movement B). This is therefore a relatively big neighbourhood, if there are x tracks and y movements between the original movement and the next movement after the original movement, there are thus $x * y$ operations to be considered. Note that y is generally a small number (the amount of movements between the two movements A and B in the ordered movement list of the POS).

We will now show a situation where this operation fixes the conflict. In the situation, which can be seen in Figure 20, train A has to move to Service Track 2 to perform a service. Train B has to move to track 3. The movement of train A is thus considered before the movement of train B in the current POS.

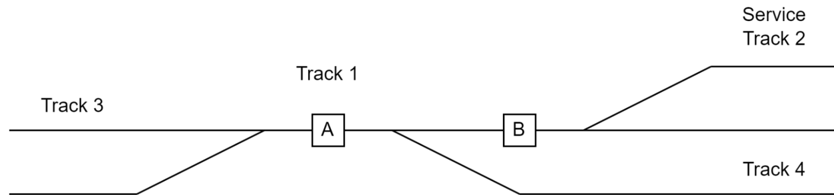

Figure 20: The original situation.

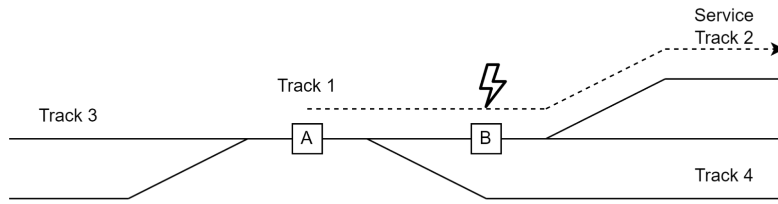

Figure 21: Train A moves to Service Track 2, causing the crossing.

Note that, in order to solve this problem, we could also consider parking train B on track 4. This neighbourhood is considered later.

Now consider the parking insert neighbourhood. When evaluating the neighbourhood, at some point we will consider moving train A to track 4, and insert the movement to this track before the movement of train B to track 3. If we now re-consider the original situation and evaluate what happens, the following happens if we consider this particular candidate:

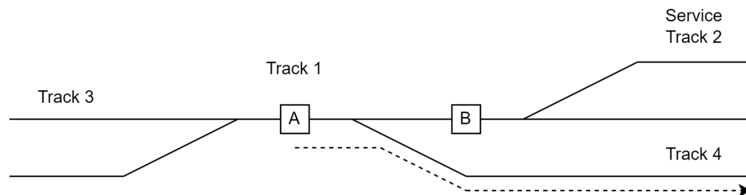

Figure 22: We first move train A to track 4.

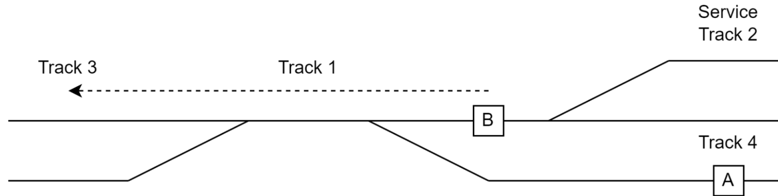

Figure 23: Now, we move train B to track 3.

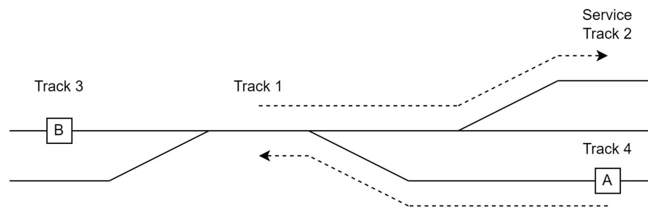

Figure 24: Then, we move train A to the original destination Service Track 2 from track 4.

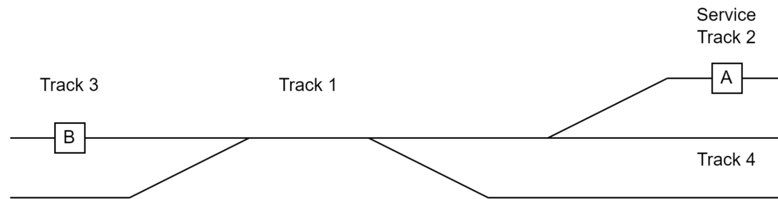

Figure 25: The final situation, which is now conflict-free.

Therefore, in this particular situation, if we consider the parking insert neighbourhood, then in this case there is thus a candidate which removes the conflict.

Movement merge

Another operation which is considered is merging two movements. This takes two movements (which in this case are thus the two movements related to the active train) and merges these movements into one. Therefore, the train thus originally moved from a track T to the origin track, and then it moved from the origin track to the destination track D, where thus the second movement caused the crossing conflict. After executing this operation, the train moves from track T immediately to the destination track D.

In order to visualize the situation, we create a situation where there are actually two crossing conflicts. See the starting situation in Figure 26.

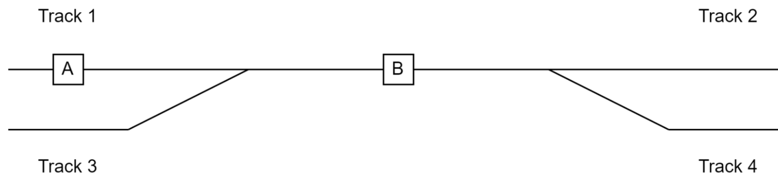

Figure 26: The starting situation.

The train A now first moves to track 2, parks there, and then moves to track 3. This results in two crossing conflicts:

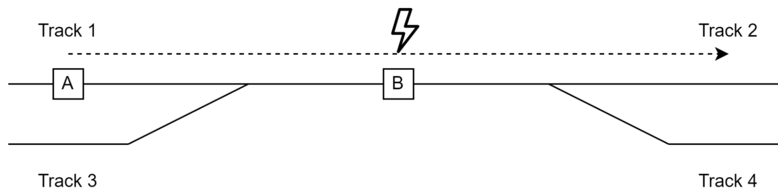

Figure 27: The first crossing conflict.

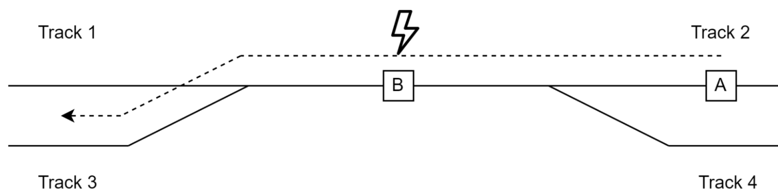

Figure 28: The second crossing conflict.

Now consider the first conflict. If we execute the movement merge operation here, then the result is thus merging the two movement operations, such that the train moves from track 1 to track 3 directly. This thus immediately solves both crossing conflicts. In Figure 29, the situation is shown after executing this operation.

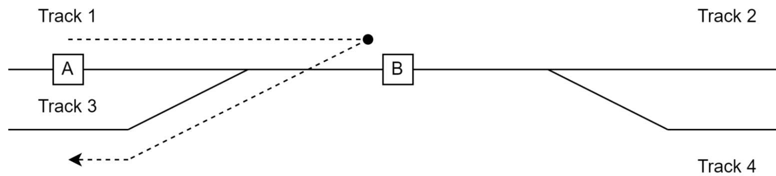

Figure 29: The situation after the movement merge situation, with no conflicts.

Note, that if we would consider the second conflict, in this iteration of our algorithm, the parking change operation is also considered. In this case, one of the neighbours of this operation is switching the destination of the movement from track 3 to track 4 (the origin of the movement is thus track 2). Hence, the movement will thus change to moving the train from track 2 to track 4, resulting in one less crossing conflict. This node is thus inserted in our search tree. However, since all conflicts in our algorithm are evaluated, the movement merge operation actually solves two conflicts, thus has a lower score than the node inserted by using the parking change operation. Hence, this node is explored first. Of course, at some point, the algorithm might backtrack to the parking change node if this is, at that point, the unvisited node with the lowest conflict score.

The passive train

Here, we will describe the operations which are considered to be operated on the passive train. Note, that for the passive train, there are two interesting movements. The passive train is parked at time of the crossing conflict at track T. There is thus a movement which has this track as destination, and there is also a movement originating from this track. These two movements are considered in the following operations:

Parking insert

This neighbourhood works the same as the parking insert neighbourhood on the active train. The passive train is currently parked on track T. It moved from some track Q to track T, and will later move from track T to some track Z.

However, in this case, we thus insert a movement in the movement from a track Q to track T, or insert a movement in the movement from track T to track Z.

Just like with the active train, this situation can be thought of "temporarily" parking the passive train on another track. This situation thus might create an opportunity for the active train to move over track T, without causing a crossing conflict, since the passive train is thus now parked elsewhere.

Parking change

Just like with the active train, we can also decide to park the passive train on

another track than T. If this is possible without causing more conflicts, the track T is thus free and the active train can move over it without any problems.

5.2.2 Track length violation

A track length violation occurs if the total length of the trains parked at a track T exceeds the track length at any given point in time. A natural way to solve this problem is to move any of these trains to another track. Note that also in this conflict, for each train participating in the conflict, there are two interesting movements. The first movement is the movement from any track to track T and the other movement is the movement from track T to another track. Note that for the figures in this section, the box size of the train represent the length of the train: if the boxes together do not fit on a track, this is thus a track length violation conflict.

For each train participating in this conflict, we consider the following operations:

Parking change
This is a very logical operation. Instead of parking any of the participating trains at track T, we simply park it elsewhere. In the figures below we show this operation:

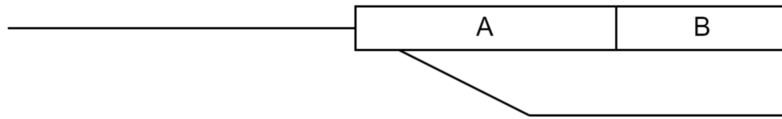

Figure 30: The original situation, where at some point both train A and train B are parked at the same track, which combined length is larger than the track length.

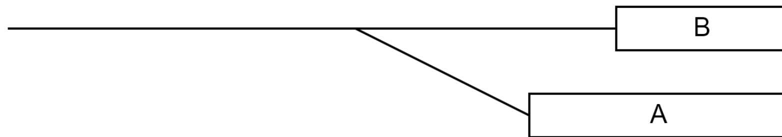

Figure 31: If train A is parked at another location, this solves the problem.

Parking swap
In this operation, we take two trains and swap their parking location. It can thus be thought of as operating the parking change operation twice, very specifically on those two trains and thus swapping the tracks they are parked on. In the figures below we show a situation where this operation solves the conflict.

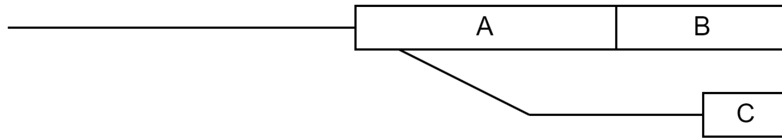

Figure 32: The original situation, where train A and train B are parked at the same track, which combined length is larger than the track length.

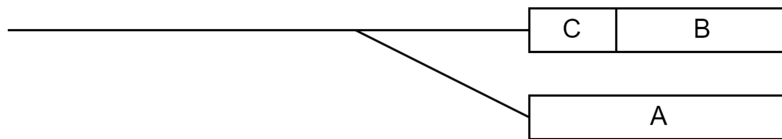

Figure 33: If train A and train C swap their parking locations, the conflict is solved.

Parking insert

This operation again works the same as described before. We thus take either of the two considered movements for each train participating in the conflict and insert a movement operation into the POS. It is however not very intuitive to see why this would solve a conflict. If the train is temporarily parked at another track, but then still moves to the conflict track T, then it be somewhat logical to conclude that this does not solve the track length violation conflict. However, below we give a situation where this operation does solve the conflict.

In this situation, there are three trains participating: trains A, B, and C. In Figure 34, the starting situation of the shunting yard is shown.

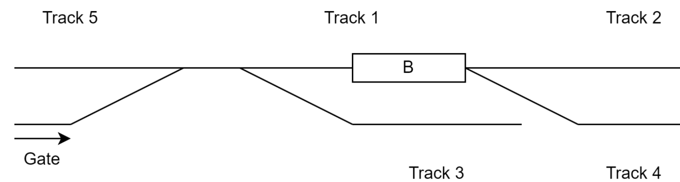

Figure 34: The shunting yard.

The movements considered in this conflict are (in this order):

- Train A to track 1
- Train B to track 2

- Train C to track 3
- Train A to the gate

Note that this is a partial POS: only the relevant movements are shown here in order to describe the conflict and how to solve the conflict. In the figures below, the movements above are visualized.

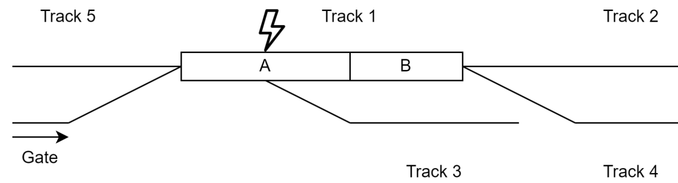

Figure 35: Train A moves to track 1, causing the track length violation conflict.

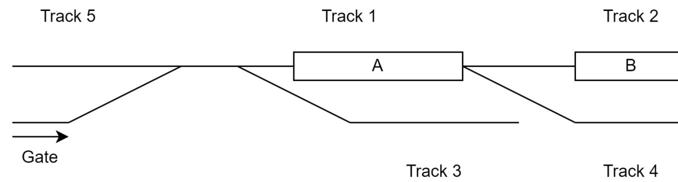

Figure 36: Train B moves to track 2.

Note: only a partial POS is shown. It is assumed that for nonzero time both train A and B occupy track 1. This is due to the precedence relations in the POS. For instance, the routes to track 2 might be blocked for train B, or it is doing a service task. We continue below with the visualizations.

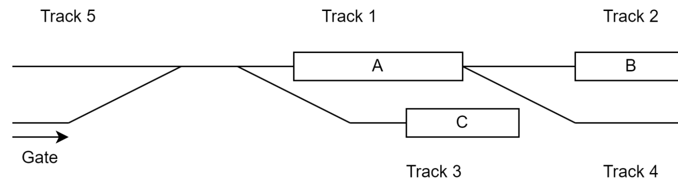

Figure 37: Train C moves to track 3.

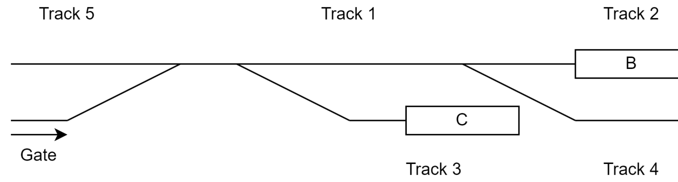

Figure 38: Train A moves to the gate.

In order to solve the conflict, consider inserting a movement for train A, where before this train is moved to track 1, it is parked at track 3. We have to insert the movement at some place in the ordered movement list. Consider the order below:

- Train A to track 3
- Train B to track 2
- Train A to track 1
- Train C to track 3
- Train A to the gate

If we now evaluate this (starting from the starting situation, see Figure 34), we get the following situation:

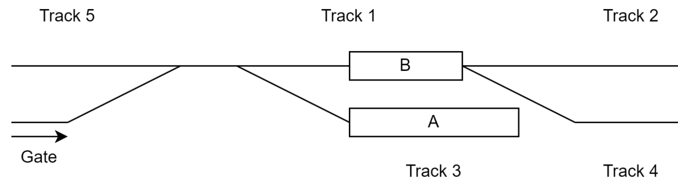

Figure 39: Train A moves to track 3.

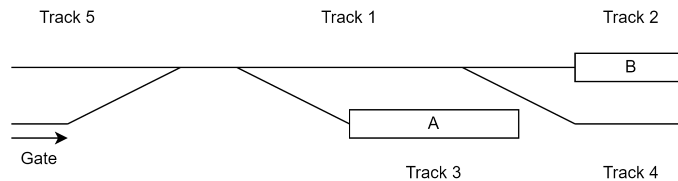

Figure 40: Train B moves to track 2.

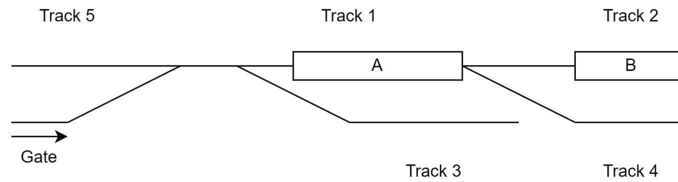

Figure 41: Train A moves to track 1.

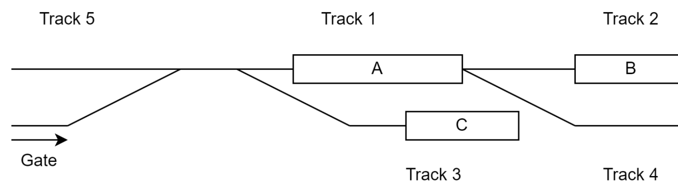

Figure 42: Train C moves to track 3.

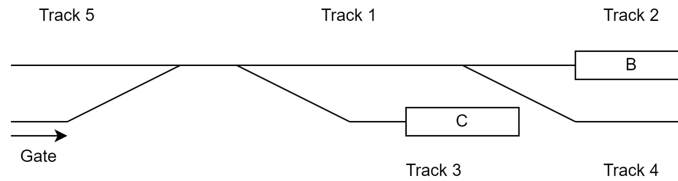

Figure 43: Train A moves to the gate.

This thus solves the conflict. Note that this conflict would not be solved if we would have done a parking change operation on train A: this would move train A to track 3, and subsequently (due to the movements schedule) move train C also to track 3. This would result in either a track length violation, or in a crossing (train A has to move through C in order to reach the gate), or both. Note that train A cannot be parked at track 5, because this track is too small for the train. However, train A can use track 5 and the small track between track 5 and track 1 in order to perform a saw movement to move from track 3 to track 1.

5.2.3 Arrival Delay

As a recap: an arrival delay happens if a train cannot enter the shunting yard at the scheduled time. This could have to do with the disturbed train, but it could

also happen because of the changed conditions at the shunting yard. These conflicts arise, because the conflicted train A (this could be the disturbed train which arrived later, but it could also be another train) arrives at the gate track at the scheduled time (unless it is the disturbed train, which now of course arrives at a later time than originally scheduled), and then has a movement activity which has a certain destination track T. However, due to another train B using part of the track which the train has to use to arrive at track T, it is thus not allowed to move on these tracks, since these tracks are blocked. Note that this conflict thus arises by the way how HIP evaluates the routes. In this case, the movement of B is considered before the movement of A is considered. Therefore, B blocks the arriving train A from entering the yard. However, this conflict would not arise if the movement of A is considered before the movement of B: in that case, B does not block the route when calculating the earliest starting time of the movement of A. Note that this would likely schedule the movement of B later, and this could cause other conflicts when evaluating the POS.

Therefore, there are two interesting movements regarding this conflict. The first is obviously the movement of the arriving train A. The other movement, is thus the movement of train B which blocks the arriving train to enter the shunting yard.

Below, we explain what neighbourhoods are considered.

Parking change and parking insert

Note that due to how HIP evaluates the POS, if we have a movement, the entire route has to be clear for the entire duration of the movement. If we make the movement shorter in time (either the movement of the arriving train or the blocking train), the movement blockage time is thus shorter and might either solve the conflict, or reduce the arrival delay time (reducing the conflict costs). To make the movement time shorter, consider the parking change and parking insert operations. These will both move a train to another track, possibly reducing their movement time. Since the target track is changed, it is also possible that the route to this track now does not use a track piece which is used by both trains, which would immediately solve the conflict, since now the track is not blocked anymore. Below we give an example how the conflict is solved using a parking change operation.

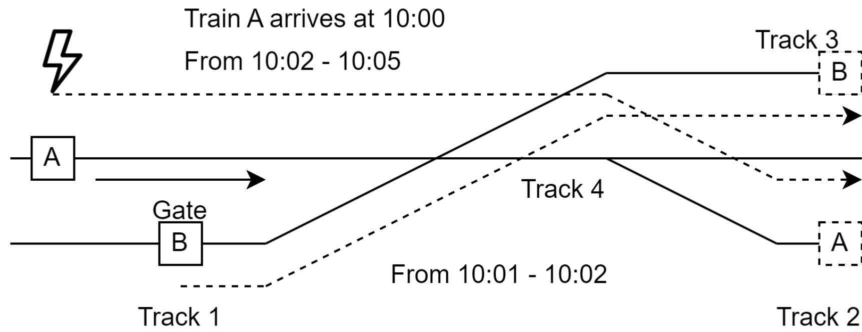

Figure 44: The conflict situation: train A arrives at the shunting yard at 10:00, but can only enter the yard at 10:02. The movement from the gate to track 2 takes 3 minutes. Train B moves from 10:01 - 10:02 from track 1 to track 3.

Instead, if we move train A to track 4 instead of track 2 (this movement takes only 1 minute):

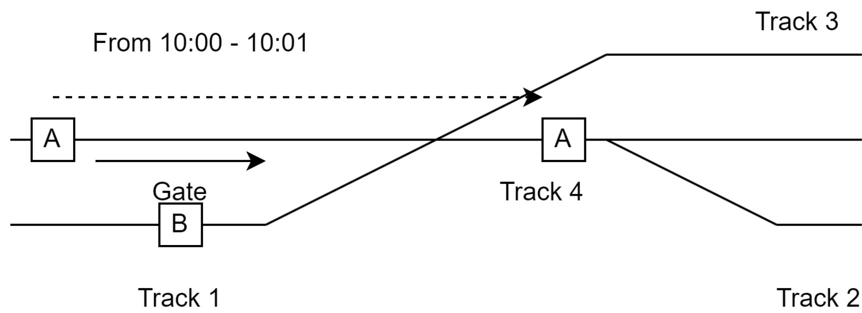

Figure 45: Since the route is not blocked, train A can now drive from the gate to track 4 and thus enters the yard at the schedule time

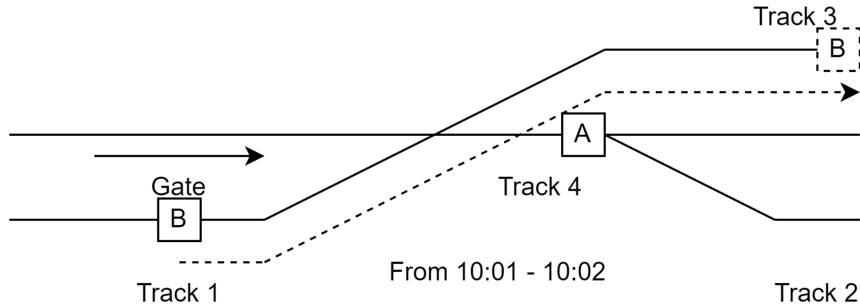

Figure 46: Train B moves, just as before, from track 1 to track 3

Movement switch
In this operation, the order in which the movements are evaluated is changed. Consider the same situation as in Figure 44. In the original situation, the movement of train B is evaluated first, and then the movement of train A is considered. Note that HIP thus schedules the earliest time at which movements can take place. In the original situation, B thus moves at 10:01.

Now consider evaluating the movement of train A first. It thus moves from the gate to track 2, which takes 3 minutes. If we now calculate the earliest time which we can move the train, this is thus 10:00. Train A now thus uses the route from 10:00 - 10:03. If we now evaluate the movement of train B, it will thus now move from 10:03 - 10:04 to track 3. Note that although this might solve the arrival delay conflict, this could yield other conflicts caused by B now moving to the track later.

5.2.4 Departure delay

A departure delay happens if a train A leaves the shunting yard later than the scheduled departure time. This conflict gives the most freedom in changing the POS: any movement which the train makes, can be changed (of course, only if this is within the allowed time window where it is permitted to make changes to the POS). The conflict is caused, because the train has to perform too many tasks on the shunting yard, and/or has to wait too much time before performing any of these tasks. It thus makes sense to solve this conflict by trying to reduce the delay time between tasks, or remove certain tasks (such as movements). Note that even though a single operation does not solve the conflict, it might still reduce the departure time, thus decreasing the conflict score which is a better situation.

Note that in the end this conflict is thus caused by any of the tasks which the train had to do took too long. It might be that the departure train was at a combine/split operation at the yard. Therefore, it makes sense to look at all the relevant activities of all the train units of this departing train. If we manage

to combine/split the train earlier after performing an operation, then this time saving might propagate through the POS, possibly solving or decreasing the departure delay conflict.

We will describe the considered operations and give some examples how these might solve these conflicts.

Movement shift

In this operation, we take a movement operation related to one of the relevant train units, and move it to another place in the movement activities list, such that it is either evaluated earlier or later when evaluating the POS. Recall that when evaluating the POS, the order of the movements determines the situation at the shunting yard when calculating the routes. Therefore, when evaluating a movement earlier or later, the positions of all the trains at the shunting yard are different, and therefore can result in the movement activity being scheduled earlier, and therefore decrease the conflict costs or solve the departure delay conflict.

In one situation where this operation solves the conflict immediately, is related to how we solved the arrival delay conflict. For instance, if the gate where the train should depart from is blocked by another movement, we can solve this in the same way as when a train arrived at the shunting yard, but the route from the gate to the target track is blocked by another movement. To see this, refer to the situation in Figure 44, and execute the operations in "reverse": consider all the movements in decreasing order, the target tracks are now the origin tracks and the origin tracks are now the target tracks.

Movement merge

To solve a departure delay conflict, it makes sense to reduce the amount of movements of the conflicting train. This ensures that the train does less movements on the yard. It is therefore possible that due to reduced movements, all the tasks on the shunting yard can now be done faster such that the train can leave on time. We show a situation which solves the conflict below.

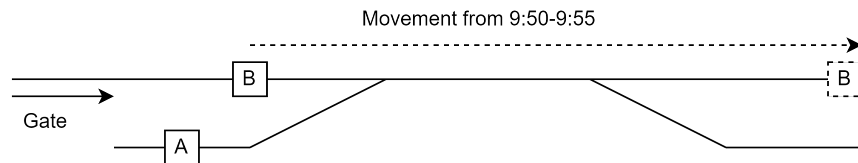

Figure 47: Train A has departure time 10:00. Train B moves first.

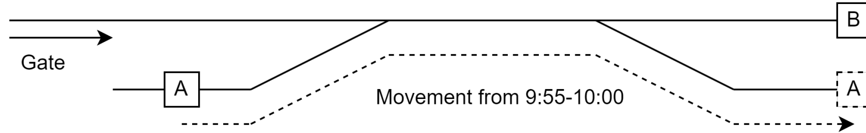

Figure 48: Due to the blockage of the route of train B, train A can only start performing this movement at 9:55.

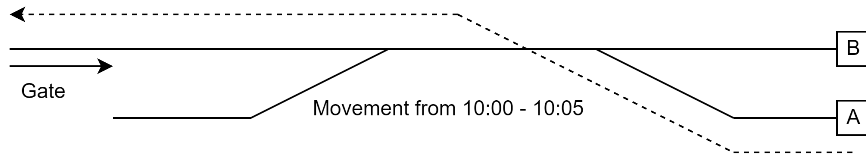

Figure 49: After reaching the target track, train A moves to the gate, but only departs at 10:05, so 5 minutes late.

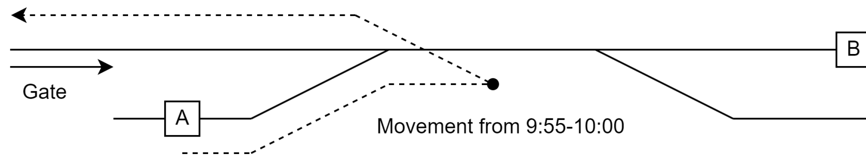

Figure 50: Consider merging the movements of train A from Figures 48 and 49. Since the track where train B was parked first is free from 9:55, it is now possible to perform the movement from the original track to the gate from 9:55 - 10:00, thus departing exactly on time.

Parking insert and parking change
For all the movements on the shunting yard of the considered train units (thus the ones involved in the conflict), parking change operations are considered. By changing the destination track of these movements, it is possible that subsequent tasks can be executed earlier. The parking insert operation is only considered for the movement right before the departure, for the same reasons as why it is used in the arrival delay conflict. We do not want to start inserting movements for the departure delay conflict, since this only adds tasks to the conflicting train (which thus departed late), which most likely increases the time the train stays on the yard and therefore increases the conflict cost. However, there are situations thinkable where this would actually decrease the costs.

Service task order swap
Recall that one of the precedence relations in the POS has to do with the service tasks. Before a service task can be started, the previous service tasks have to be finished. If we change the precedence relations of these service tasks, this might solve the conflict, by starting one service task earlier and another task later.

Below, we show an example situation where this solves the conflict.

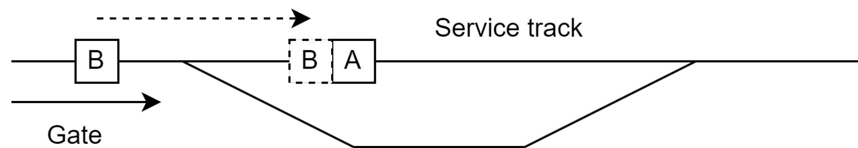

Figure 51: In this situation, train A and B are both scheduled a service task at the service track. Train B has to finish the service task first and then train A is serviced. Train A is scheduled to leave the yard at 10:00. Train B arrives at the service track at 9:50. Train A is at the service track since 9:40.

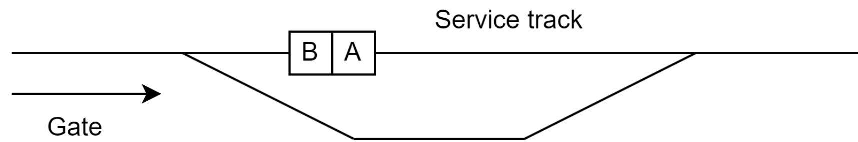

Figure 52: The service tasks are performed. Train B is serviced from 9:50 - 10:00 and train A is serviced from 10:00 - 10:10.

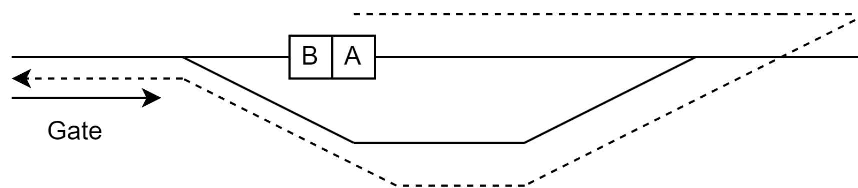

Figure 53: Train A now leaves the shunting yard, performing the final movement from 10:10 - 10:17, thus having a departure delay of 17 minutes.

Now consider swapping the order of the service tasks. In this case, train A is now serviced from 9:40 - 9:50. Train B is subsequently serviced from 9:50 - 10:00. When evaluating the departing movement of train A, it will now perform

the final movement from 9:53 - 10:00, thus departing on time, resolving the conflict.

Matching swap

The final, complex operation neighbourhood is the matching swap operator. This operator takes two departing trains A and B which have identical units. It swaps the evaluation of the movements of these two trains around after a "pivot point". Therefore, the movements of A are now evaluated after the movements of B, and the movements of B are now evaluated before the movements of A. It also swaps the departure time of the two trains. In this neighbourhood, we take the train which has a departure delay as train A, and thus try to swap it with any other train B with the same units.

To show how this works in practice, take the following example:

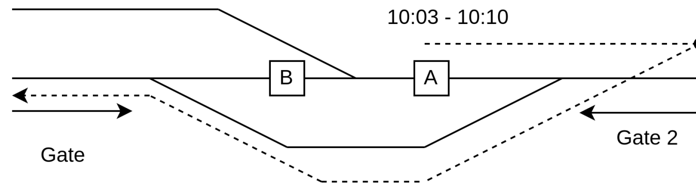

Figure 54: In this situation, both train A and train B have to depart at 10:10. Train A has to depart from the Gate, and train B has to depart from Gate 2. First, we evaluate the movement of train A and then the movement of train B. To avoid a crossing, train A has to travel a long route to reach the gate. It uses this route from 10:03 - 10:10.

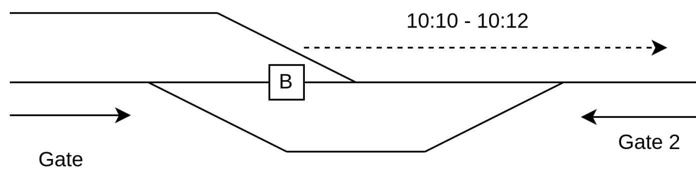

Figure 55: Since the route which train B has to use is reserved till 10:10, it can only depart from the track at 10:10. The travel takes 2 minutes, so it will arrive at Gate 2 at 10:12, a departure delay of 2 minutes.

If we now consider the matching swap operation, this thus means that the movement of train B is evaluated first, and then the movement of train A. Note that the departure tracks of the trains are also swapped: this means that train B has to depart from the Gate, and train A has to departure from Gate 2. If we perform this operation:

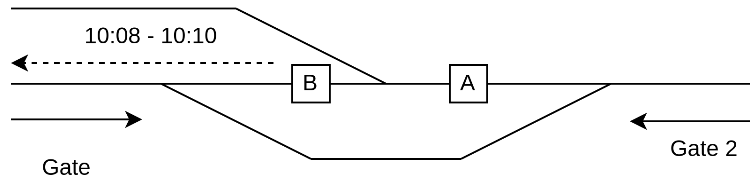

Figure 56: We first consider the movement of train B. It will now move to the Gate from 10:08 - 10:10.

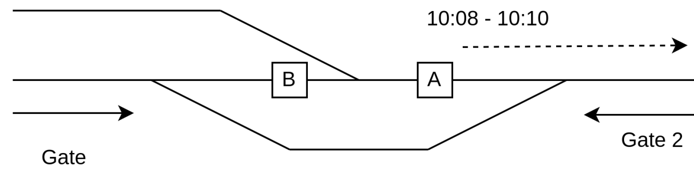

Figure 57: Since the track is now not blocked, train A can departure on time too: it moves from 10:08 - 10:10 to Gate 2, and the departure delay conflict is resolved.

We have now explained and motivated all neighbourhoods considered per conflict. We will conclude this chapter by explaining how we check the feasibility of the resulting schedule.

Checking feasibility regarding the shunting yard disturbance constraints

Due to the complex method of evaluating a POS and assigning routes and starting times to each activity, each time an operation is executed, the resulting POS is checked whether it does not violate the constraint of the shunting yard disturbance constraints: it is not possible to change an activity before time $\mathbf{X} - \mathbf{A}$. If any of the activities before time $\mathbf{X} - \mathbf{A}$ changes, then we do not consider this operation. It is not trivial to know before executing an operation if this will violate the constraint, so this is evaluated after execution.

6 Experimental setup

In order to test this algorithm, we test this algorithm on two datasets: one dataset with shunting problems at the Kleine Binckhorst, and one dataset with shunting problems at the Grote Binckhorst. The Kleine Binckhorst is a carousel-like yard, and the Grote Binckhorst is a shuffleboard-like yard. This ensures that the algorithm is not optimized for a specific layout of a shunting yard.

6.1 Generating the problems

For each yard, 999 shunting problems are generated. For Kleine Binckhorst, there are 333 scenarios with 17 train units, 333 scenarios with 18 train units and 333 scenarios with 19 train units. For Grote Binckhorst, there are 333 scenarios with 36 train units, 333 scenarios with 37 train units and 333 scenarios with 38 train units. These numbers were chosen such that these are close to the "capacity" of the shunting yard: there have been experiments at NS which used HIP to find how much train units can use the shunting yard at the same time where HIP finds a solution within a reasonable time. In order to generate these problems, an existing tool is used, which generates these problems. To generate realistic situations, statistical data is fed into this generator. This includes:

- The ratio of the train compositions in real life. For example, if 10% of the trains consist of 1 SLT-4 and 1 SLT-6 unit, this will be reflected in the generated problem instances.
- The distribution of the service tasks. For instance, certain safety controls have to be done every 2 days, while heavier safety controls are done every 14 days. This frequency depends on the specific train type.
- The distribution of the arrival/departure times. In the current input, first all trains will arrive, and then they will depart. This is the result of the statistical analysis.
- For the shunting yards which we use, only a single arrival/departure track is used. This does reflect reality: the other tracks are sporadically used. For Kleine Binckhorst this is track 906a and for Grote Binckhorst this is track 103b.

It should be noted that the train composition ratios are taken over all shunting yards in The Netherlands, so not the yards which we are interested in. These are also taken over an entire week, so not a specific day. Hence, the generated problem does reflect a "general shunting yard" problem, not a problem on a specific shunting yard.

The generated problems will have a minimum time between two arrivals or departures. The reason is that if the arrival/departure times are randomly sampled, it is possible that two arrival/departure are so close to each other that due to the constrains in HIP, the problem is not solvable. For instance, in the extreme case that two trains arrive at exactly the same train, then one of these trains will block the track, and therefore there will always be an arrival delay conflict.

These generated problems are solved in HIP, with a maximum runtime of 1 minute per problem. This results in 901 feasible schedules for Kleine Binckhorst and 828 feasible schedules for Grote Binckhorst.

In our algorithm, only these feasible schedules are considered, for which we will first generate a disturbance and then try to solve this using either HIP or our algorithm.

The feasible schedules are thus generated by HIP, where it could only take a minute to solve the problem. This time is less than the default time of HIP to solve the problem. Therefore, the feasible plans which we give as input to our experiment might be relatively "easy" problems to solve, and might therefore also be more robust against disturbances than problems which HIP would take longer to solve.

In order to generate disturbances, the following procedure is taken:

- Take a random arriving train
- Schedule the train to arrive **T** minutes later, where **T** is randomly picked between 1 and 60 minutes
- Evaluate the POS found by HIP. If there are nonzero conflict costs, a disturbance is found. If there are still zero conflict costs, repeat the procedure.

For each disturbance, an extra simple check is taken in order to be sure that it is not trivially unsolvable. For example, consider a feasible POS where train A arrives at 10:00 and train B arrives at 10:30. These both arrive at the same gate. If we now disturb A by 30 minutes, it means that A arrives at exactly the same time as B. Since each train has to enter the yard and thus move from the gate to a track, this thus blocks the track originating from the gate. Since either A or B will block the gate track, this thus results in an arrival delay conflict which cannot be solved. Hence, this is related to the checks done by the problem generator mentioned earlier.

These disturbances are saved and are now ready to be solved using either HIP or our algorithm. Therefore, two datasets now exist, with disturbances at both the Kleine Binckhorst and the Grote Binckhorst.

6.2 Running the experiments

For each problem in each of the two datasets, evaluate the problem using both HIP and our algorithm. There is a limit of 10 seconds for both HIP and our algorithm. We choose $\mathbf{A = 15}$ **minutes**. It is therefore possible to change the POS 15 minutes before the disturbed train was originally scheduled to arrive at the yard.

For both HIP and our algorithm, we report the depth. For HIP, this is the amount of operations done on the POS. Sometimes, HIP also backtracks, and therefore we decrease the depth of HIP in case HIP does backtrack. The depth of our algorithm is also the number of operations necessary from the root to reach the feasible POS.

7 Results

In this section, we give the results of the experiments. For Kleine Binckhorst, there were thus 901 problems and for Grote Binckhorst there were 828 scenarios.

	Solved HIP	Solved Algo	Correct HIP
Kleine Binckhorst (901)	854 (94.8%)	650 (72.1%)	269 (31.5%)
Grote Binckhorst (828)	751 (90.7%)	605 (73.1%)	211 (28.1%)

It can be seen that there are no big differences between the two different shunting yards. This thus gives the impression that the algorithm is not optimized for a specific type of shunting yard (so for instance the carousel (Kleine Binckhorst) or the shuffleboard (Grote Binckhorst)), which is a good result, since this implies that the layout of the yard does not have a huge impact on the performance. The "Correct HIP" column describes how many solutions of HIP did not violate the constraints of the disturbed problem, thus no activities before time $\mathbf{X} - \mathbf{A}$ were changed. Since HIP performs random operations, it is very well possible that HIP changes something before time $\mathbf{X} - \mathbf{A}$.

This could be prevented by adding a guard in HIP, which, after executing an operation, checks if the resulting POS violates this constraint. If this is the case, then do not consider this operation. However, this will immediately make HIP much slower: for instance, if 50% of the random operations would violate the constraint, HIP would thus be at least two times slower (and possibly more, since it now might have much more trouble finding a solution). This is therefore not a great way to compare both algorithms, since adding a "handicap" on HIP in order to compare the algorithms is not fair.

Another great result is that, although our algorithm finds less solutions than HIP, it does find solutions for problems which HIP could not solve. For the Kleine Binckhorst, our algorithm found 22 solutions (2.4% of Kleine Binckhorst problems) which HIP could not find. For Grote Binckhorst, this increased to 39 solutions (4.7% of Grote Binckhorst problems) which HIP could not find. This is one of the differences which is interesting, since it seems like our algorithm performs better on Grote Binckhorst than Kleine Binckhorst by thus finding relatively more solutions to problems which HIP could not find.

In the table below, we compare the average runtime of the algorithms. This is the average runtime considering only the solutions which the algorithms found.

	Avg. runtime HIP (ms)	Avg runtime Algo (ms)
Kleine Binckhorst	1225	670
Grote Binckhorst	1510	750

If we look at the average runtime (this is the average runtime considering only the problems which were solved), we can see that our algorithm is about 2x faster if it finds a solution. What is interesting, is that the average runtime is relatively low, which seems to imply that if a solution is not found fast, then it is never found.

To investigate this, we take a look at the depths of which the solutions were found. For our algorithm, the max depth reached is only 9 at Grote Binckhorst and 13 at Kleine Binckhorst. Below, we give a table for our algorithm and histograms for HIP.

Depth	Frequency Kleine Binckhorst	Frequency Grote Binckhorst
1	522	513
2	62	50
3	21	15
4	17	13
5	13	5
6	5	4
7	4	3
8	2	1
9	1	1
10	0	0
11	2	0
12	0	0
13	1	0

From these tables we can conclude that if a solution is found, using our algorithm, this does not need a lot of operations.

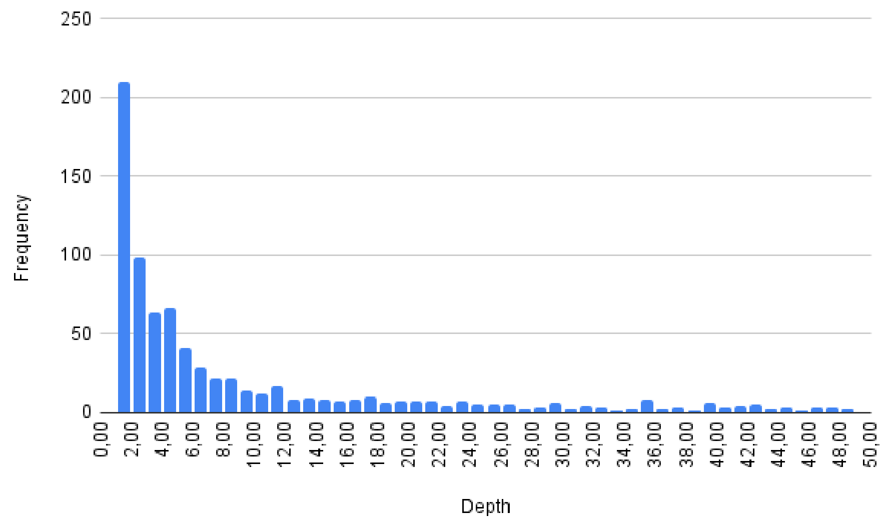

Figure 58: Depth histogram Kleine Binckhorst. Note: this histogram is cut off at depth 50. The max depth reached at Kleine Binckhorst is 192. From strictly 50 - 192, there are 88 problems which fit in that range.

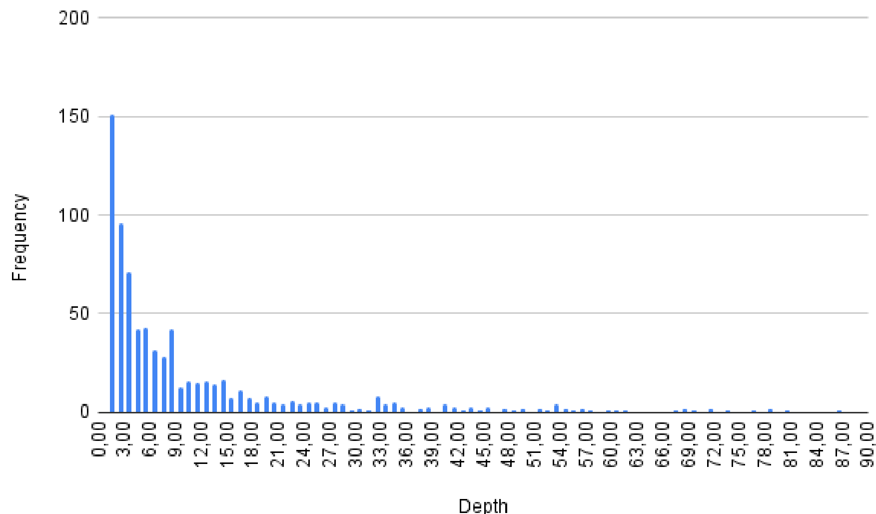

Figure 59: Depth histogram Grote Binckhorst. This histogram is not cut off. The max depth is 86.

For HIP, we also see that most solutions are almost immediately found. If HIP has to search at higher depths (thus has to apply more operations), the chance that HIP finds a solution decreases. Compared to our algorithm, it seems like our algorithm is “smarter” by finding more solutions at depth 1. This is however an unfair comparison: HIP takes a random neighbourhood, executes it, and then takes the best operation and continuous from there. Our algorithm takes multiple neighbourhoods, executes them all and picks the best operation. Of course, if at any point a feasible schedule is found, the algorithm quits immediately and reports the result.

One of the reasons HIP might find more solutions in total than our algorithm is that it will keep doing random operations (and sometimes backtracking). It is possible for our algorithm to escape a local minimum, but this is less likely. If HIP gets “lucky”, it might use a neighbourhood escaping this local minimum and thus entering a new situation where it is more likely it finds a solution. HIP thus allows for changes before the time **X - A** and therefore has more freedom to solving the disturbed problem than our algorithm.

Results without the time constraint

If we let our algorithm run without the **X - A** constraint (all activities can be edited), we can directly compare our algorithm to HIP. The results are that for the Kleine Binckhorst, 801 (88.9%) of the problems were solved (compared to 854 (94.8%) of HIP). Out of these, 29 problems are solved which HIP did not solve. These solved problems include the 22 problems which were solved with

the **X - A** constraint.

For the Grote Binckhorst, 733 (88.5%) of the problems were solved, compared to 752 (90.7%) of the problems solved by HIP. Out of these, 50 problems were not solved by HIP. 12 of these problems were not solved with our algorithm with the **X - A** constraint in place. 1 problem was solved with the **X - A** constraint in place: this problem was not solved if we removed the **X - A** constraint.

It can thus be seen that if we remove the constraint, the results are closer to the results of HIP. This shows that the chosen neighbourhoods are not perfect, but they are close to finding quick solutions. It also shows that if we impose the **X - A** constraint, this significantly reduces the amount of solutions which we have found if we do not impose this constraint, thus this makes the problems harder.

7.1 Unresolved problems

An interesting question is: if HIP solves a problem, but our algorithm does not, what is the best conflict score our algorithm achieves and how many times does it stay stuck at this exact situation? In this small section we only consider the count of conflicts, so not the variable costs (i.e. the time it takes to resolve most conflicts). There are many different combinations of the conflicts, but for both the shunting yards we will give the top-5 recurring conflicts. Note: AD are Arrival Delays, DD are Departure Delays, CR are Crossings and TLV are Track Length Violations.

Frequency	AD	DD	CR	TLV	Frequency	AD	DD	CR	TLV
57	0	0	1	0	56	1	0	0	0
30	1	0	0	1	21	2	0	0	0
15	2	0	0	1	14	0	0	1	0
14	0	0	0	1	12	0	0	0	1
10	0	0	1	1	8	3	0	0	0

Table 1: Top - 5 conflicts for both yards (left: Kleine Binckhorst, right: Grote Binckhorst). There are thus 57 unsolved problems where there is only one crossing conflict left on the Kleine Binckhorst.

An interesting result is that for the Kleine Binckhorst (carrousel), a crossing or an arrival delay conflict seems to be a bottleneck, while for Grote Binckhorst (shuffle board), only the arrival delay conflict seems to be the big bottleneck.

It should also be noted that all the problems are solved if we give HIP a longer runtime. The longest runtime recorded to solve a problem is 158 seconds.

8 Conclusion

Although the algorithm does not solve more situations than HIP, the algorithm does solve some situations which HIP does not solve. This is an indication that this is an interesting area of further research. Since these solutions are found relatively quickly, it would make sense to sometimes run our algorithm for one (or maybe two) iterations in order to see if there is an operation in the neighbourhood regarding the current conflicts which solves the problem.

We have compared the results of HIP against our algorithm. However, since HIP has more freedom in solving the problems (it is allowed to make changes before time **X - A**), it is therefore not very surprising that HIP solves more problem than our algorithm does. It is possible to edit HIP in order to include the rule that it cannot makes changes before time **X - A**, but this is a very complex operation which is not considered in this thesis. Our algorithm thus solves problems which HIP does not solve in the time frame of 10 seconds, which is a nice result. Therefore, if HIP and our algorithm are run in parallel, the end result is that more problems are solved in total than if only one of the algorithms is picked.

9 Discussion and further research

In this section we reflect upon the work and propose a few topics to explore.

Smarter parking neighbourhoods
The first note we want to make is that the parking neighbourhoods are currently rather big. We essentially take a conflict which takes a parking neighbourhood and then consider parking the train on all other available tracks. It would be logical to rank the tracks to move to. For instance, a snapshot could be taken of the situation right when the route of the train is evaluated. By taking a look at this snapshot, we might find that some tracks are not interesting to move to, since they will either not solve the conflict, or possibly create new conflicts.

Non-constant crossing costs
The helpful property of the considered conflicts except the crossing conflict is that if the conflict time is reduced, the total conflict costs are reduced as well. This thus indicates this improves the solution. For crossings, however, the conflict is either solved or not. If there would be a way to start scoring certain crossing conflicts (possibly by looking at the situation when the crossing happens) and start adding a variable cost element, this might help to find better candidates, or help picking a better situation. This could improve the situation at the Kleine Binckhorst, where (compared to HIP) a lot of crossings are not solved where HIP does solve it.

Comparing solved HIP problems against unsolved problems of our algorithm
Clearly, HIP solves more problems than our algorithm. It would be interesting to see if there are cases of HIP where the problem is solved without making changes before time $\mathbf{X} - \mathbf{A}$, but our algorithm does not solve the problem.

Integration with HIP
Notice that although we have focused on the disturbance problem, our algorithm can be directly used by HIP. If we simply set the time $\mathbf{X} - \mathbf{A}$ to 0, then the algorithm has all the freedom to edit the entire POS. It might be interesting to see if this helps HIP solve some problems, especially complex problems where HIP is stuck at a final conflict for a long time. It might also be interesting to see if our algorithm helps finding solutions in nearly identical situations. Also, if our algorithm helps finding solutions faster, then this is also an improvement.

Robustness of HIP solutions
As a final remark, we note that although our algorithm only considers a small subset of the neighbours which HIP considers, it still manages to find a lot of solutions. This signals that most of the original solutions to the problems given by HIP (before applying the disturbance) are thus rather robust, where it is relatively easy to solve the conflicts.

10 References

References

[BHA18] R. van den Broek, H. Hoogeveen, and M. van den Akker. “How to Measure the Robustness of Shunting Plans”. In: *18th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2018)*. Edited by Ralf Borndörfer and Sabine Storandt. Volume 65. OpenAccess Series in Informatics (OASIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, 3:1–3:13. ISBN: 978-3-95977-096-5. DOI: 10.4230/OASIcs.ATMOS.2018.3. URL: http://drops.dagstuhl.de/opus/volltexte/2018/9708.

[Bro16] R. van den Broek. “Train Shunting and Service Scheduling: an integrated local search approach”. Master’s thesis. Universiteit Utrecht, 2016. URL: https://studenttheses.uu.nl/handle/20.500.12932/24118.

[Bro+22] R. van den Broek, H. Hoogeveen, M. van den Akker, and B. Huisman. “A Local Search Algorithm for Train Unit Shunting with Service Scheduling”. In: *Transportation Science* 56.1 (2022), pages 141–161. DOI: 10.1287/trsc.2021.1090. eprint: https://doi.org/10.1287/trsc.2021.1090. URL: https://doi.org/10.1287/trsc.2021.1090.

[Cac+14] V. Cacchiani, D. Huisman, M. Kidd, L. Kroon, P. Toth, L. Veelenturf, and J. Wagenaar. “An overview of recovery models and algorithms for real-time railway rescheduling”. In: *Transportation Research Part B: Methodological* 63 (2014), pages 15–37. ISSN: 0191-2615. DOI: https://doi.org/10.1016/j.trb.2014.01.009. URL: https://www.sciencedirect.com/science/article/pii/S0191261514000198.

[Fre+05] R. Freling, R. M. Lentink, L. G. Kroon, and D. Huisman. “Shunting of Passenger Train Units in a Railway Station”. In: *Transportation Science* 39.2 (2005), pages 261–272. ISSN: 00411655, 15265447. URL: http://www.jstor.org/stable/25769246 (visited on 04/24/2022).

[Kra12] J. Törnquist Krasemann. “Design of an effective algorithm for fast response to the re-scheduling of railway traffic during disturbances”. In: *Transportation Research Part C: Emerging Technologies* 20.1 (2012), 62 – 78. DOI: 10.1016/j.trc.2010.12.004.

[Len06] R. Lentink. “Algorithmic Decision Support for Shunt Planning”. PhD thesis. Erasmus University Rotterdam, Feb. 2006. URL: http://hdl.handle.net/1765/7328.

[Ono20] E.A.S. Onomiwo. "Disruption Management on Shunting Yards with Tabu Search and Simulated Annealing". Master's thesis. Universiteit Utrecht, 2020. URL: https://studenttheses.uu.nl/handle/20.500.12932/37757.

[Ste20] F. Stelmach. "Proactive-Reactive Approach for Stable Rescheduling of the Train Unit Shunting Problem". Master's thesis. TU Delft, 2020. URL: http://resolver.tudelft.nl/uuid:374fb4d1-220b-424b-a8fc-e6b3332a1397.

[Sza23] G.C. Szabó. "Scheduling mechanics on a Shunting Yard: skills, synchronization and train movements". Master's thesis. Universiteit Utrecht, 2023. URL: https://studenttheses.uu.nl/handle/20.500.12932/43524.